

AD-A190 618

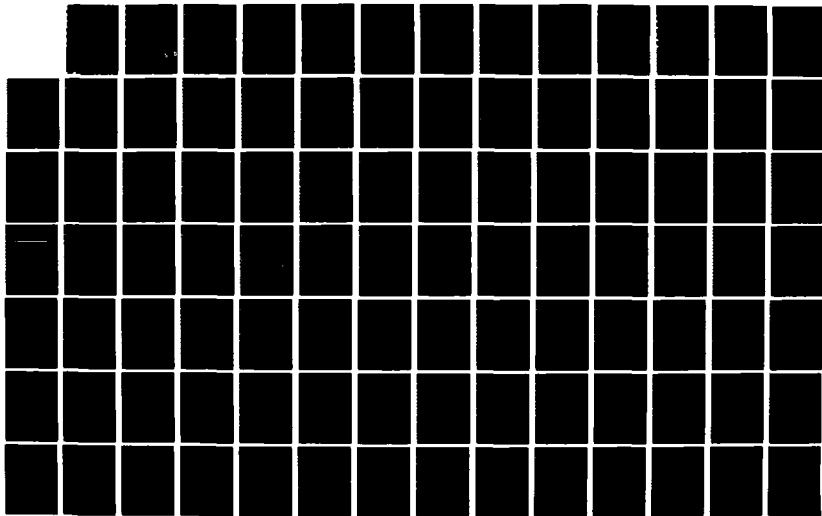
A GRAPHICS EDITOR FOR STRUCTURED ANALYSIS WITH A DATA
DICTIONARY(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON
AFB OH SCHOOL OF ENGINEERING S E JOHNSON DEC 87
AFIT/GE/ENG/87D-28

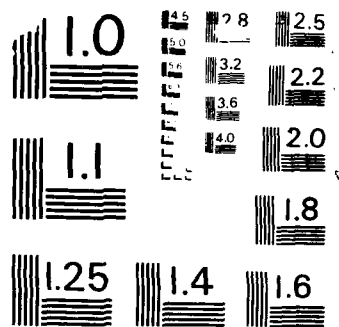
1/2

UNCLASSIFIED

F/G 12/5

ML





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A190 618

DTIC FILE COPY

1



A GRAPHICS EDITOR FOR STRUCTURED
ANALYSIS WITH A DATA DICTIONARY

THESIS

Steven E. Johnson
Captain, USAF

AFIT/GE/ENG/87D-28

DTIC
ELECTE
MAR 28 1988
S E D

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

This document has been approved
for public release and sale in
distribution is unlimited.

88 3 24 08 8

AFIT/GE/ENG/87D-28

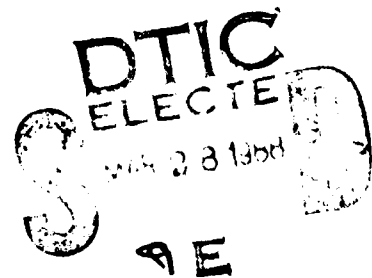
①

A GRAPHICS EDITOR FOR STRUCTURED
ANALYSIS WITH A DATA DICTIONARY

THESIS

Steven E. Johnson
Captain, USAF

AFIT/GE/ENG/87D-28



Approved for public release; distribution unlimited

A GRAPHICS EDITOR FOR STRUCTURED
ANALYSIS WITH A DATA DICTIONARY

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering



Steven E. Johnson, B.S.
Captain, USAF

December 1987

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Approved for public release; distribution unlimited

Preface

The purpose of this thesis was to build a tool that would integrate the use of structured analysis diagrams with data dictionaries for doing software requirements analysis. Separate, these two approaches are time consuming and involve a large duplication of effort. Using the tool, the duplication of effort is removed, thus improving the analyst's productivity.

The tool was designed to provide the graphical capabilities needed to create the diagram in addition to capabilities to enter the remaining data dictionary information. Data dictionary information may be transferred to and from a relational database that stores the entire project dictionary.

In preparing this thesis, I extend my gratitude to several people for their contributions. I thank my advisor, Dr. Thomas C. Hartrum, for his guidance and support for me and this thesis. Also, I thank the two other gentlemen on my committee, Major Phil Amburn and Captain James Howatt, for their expertise and assistance. Finally, I thank my dear wife, Sandy, for her understanding and moral support for me during our AFIT assignment.

Steven E. Johnson

Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	viii
Abstract	ix
I. Introduction	1-1
General Issues	1-1
Background	1-2
Statement of the Problem	1-8
Scope	1-8
Assumptions	1-8
Research Approach	1-9
Equipment and Software	1-11
Sequence of Presentation	1-11
II. Review of the Literature	2-1
Human/Computer Interface	2-1
Interfacing with a Mouse	2-7
Summary	2-9
III. Requirement Analysis for a Structured Analysis Tool	3-1
Hardware Support	3-1
Software Support	3-2
Existing Requirements from Previous Studies	3-2
Requirements for the Follow-on Tool	3-4
Summary	3-6

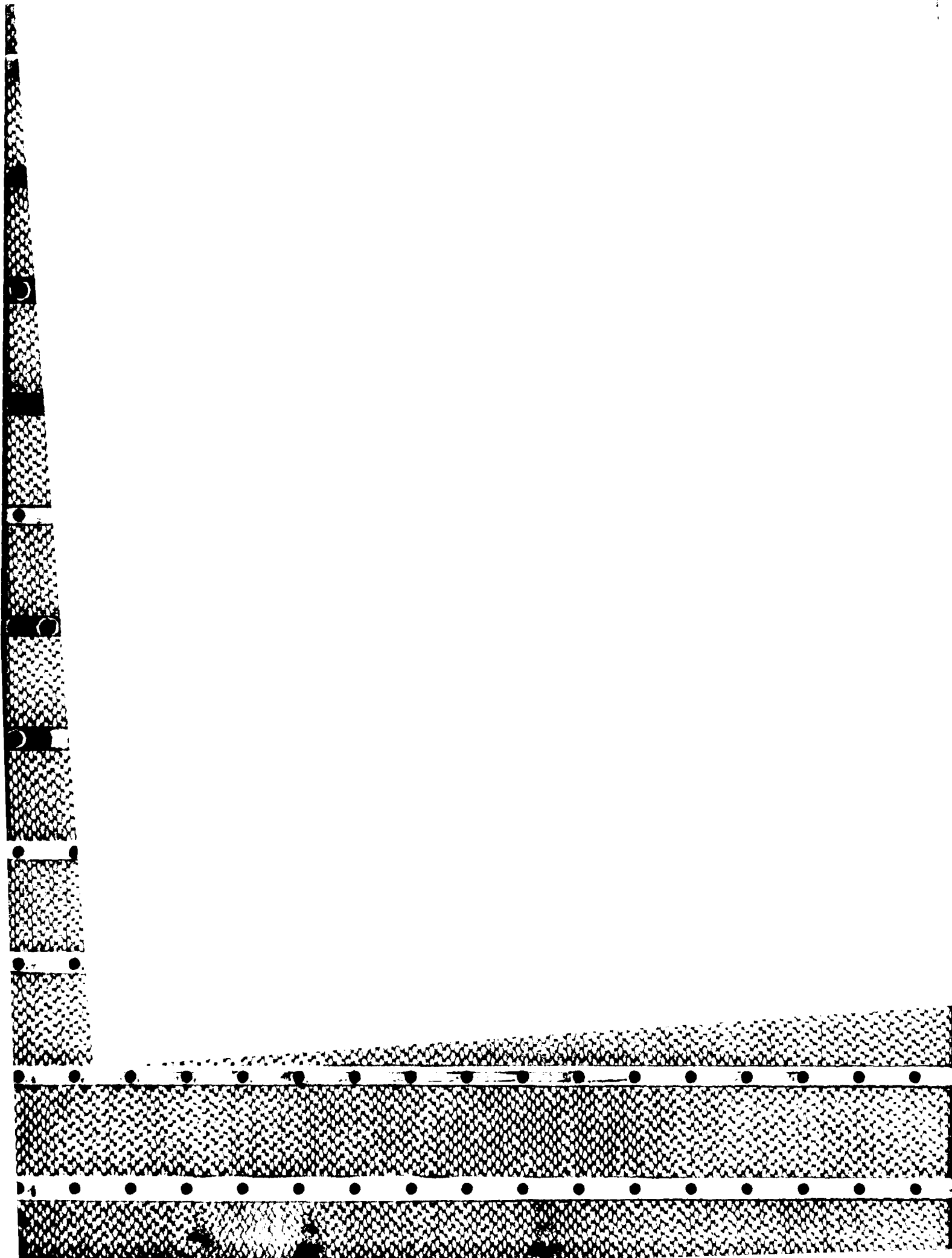
IV.	Design and Development of the SA Tool	4-1
	Hardware Considerations	4-1
	Software Considerations	4-2
	Previous Study Considerations	4-3
	Human/Computer Interface Considerations	4-3
	Screen Layout	4-4
	Menu System	4-6
	Voice Feedback	4-9
	Data Structures	4-9
	Primary Data Structures	4-10
	Data Dictionary Information Derived from the Graphics Information.	4-14
	Data Files	4-16
	Coding Approach	4-17
	Testing Approach	4-18
	Summary	4-19
V.	SA Tool Operation and Evaluation	5-1
	Operation	5-1
	Initialization	5-1
	Graphics Functions	5-2
	Data Dictionary Functions	5-4
	Facing Page Text Functions	5-5
	Input/Output Functions	5-5
	Miscellaneous Functions	5-6
	Evaluation Results	5-7
	Evaluation Methodology	5-7
	Evaluation Conditions	5-8
	Evaluation Results	5-9
	Summary	5-12
VI.	Conclusions and Recommendations	6-1
	Conclusions	6-1
	Recommendations	6-2
	Small Scale Projects	6-2
	Large Scale Projects	6-3

Appendix A: AFIT Structured Analysis Syntax . . .	A-1
SADT and IDEF ₀	A-1
AFIT Structured Analysis Diagram Syntax.	A-4
Subset of AFIT Syntax Implemented . . .	A-4
Data Dictionary Formats	A-6
Facing Page Text Format	A-11
Appendix B: Example Outputs	B-1
Appendix C: File Format Definitions	C-1
Appendix D: Configuration Guide	D-1
Appendix E: Summary Paper	E-1
Appendix F: Requirements Analysis Diagrams	F-1
Appendix G: Structure Charts	G-1
Appendix H: Thesis Software	*
Appendix I: User's Manual	I-1
Appendix J: Reference Manual	*
Bibliography	BIB-1
Vita	VITA-1

* This appendix is contained in Volume II of the thesis.
This material is maintained by the Department of Electrical
Engineering, Air Force Institute of Technology.

201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999 1000

2.	Components of the User Interface	1-4
3.	Design Principles	2-2
4.	RADD Human-Interface Requirements	2-6
5.	SA Tool Screen Layout	3-4
6.	SA Tool Menus	4-4
7.	Example Group of Lines	4-8
8.	Resulting Linked List Structure	4-12
9.	Example Evaluation Format	4-13
10.	Evaluation Questions	5-8
11.	SADT Language Features	5-10
12.	IDEF ₀ Graphic Syntax	A-2
13.	Graphic Notations Unused by IDEF ₀	A-3
14.	Implemented Graphic Syntax	A-3
15.	Format of a Data Dictionary for Activity	A-5
16.	Data Dictionary for Activity	A-7
17.	Format of a Data Dictionary for Data	A-8
18.	Data Dictionary for Data	A-9
19.	Facing Page Text Format	A-10
20.	Correctly Formatted Facing Page Text	A-11
21.	Example ".gph" File	A-12
22.	Session File Outline	C-2
23.	Example Session Header	C-5
		C-6



List of Figures

Figure	Page
1. Example SADT	1-4
2. Components of the User Interface	2-2
3. Design Principles	2-6
4. RADD Human-Interface Requirements	3-4
5. SA Tool Screen Layout	4-4
6. SA Tool Menus	4-8
7. Example Group of Lines	4-12
8. Resulting Linked List Structure	4-13
9. Example Evaluation Format	5-8
10. Evaluation Questions	5-10
11. SADT Language Features	A-2
12. IDEF ₀ Graphic Syntax	A-3
13. Graphic Notations Unused by IDEF ₀	A-3
14. Implemented Graphic Syntax	A-5
15. Format of a Data Dictionary for Activity	A-7
16. Data Dictionary for Activity	A-8
17. Format of a Data Dictionary for Data	A-9
18. Data Dictionary for Data	A-10
19. Facing Page Text Format	A-11
20. Correctly Formatted Facing Page Text	A-12
21. Example ".gph" File	C-2
22. Session File Outline	C-5
23. Example Session Header	C-6

A GRAPHICS EDITOR FOR STRUCTURED ANALYSIS WITH DATA DICTIONARY SUPPORT

I. Introduction

General Issues

This research effort was part of the continuing research in the field of software development conducted at the Air Force Institute of Technology (AFIT) by the Department of Electrical and Computer Engineering. In conjunction with this research, the department has established documentation standards to support the requirements analysis phase of the software life cycle. The documentation includes structured analysis diagrams and data dictionaries. The structured analysis diagrams use a syntax derived from Structured Analysis and Design Technique (SADT) (SADT is a trademark of SofTech, Inc.)

Two past theses led to the concept of integrating the structured analysis and data dictionary documentation standards. In 1984, an AFIT thesis by Charles W. Thomas suggested data dictionary information could be obtained from graphic pictures (Thomas, 1984:11). In 1986, an AFIT thesis by Jeffrey W. Foley developed a data dictionary editor for the Zenith Z-100 computer (Foley, 1986:3). This editor is capable of creating and editing data dictionary information

and putting the information into a database which is stored on a central computer system.

This project is a direct follow on to the 1986 thesis by James W. Urscheler. He created an initial version of a tool (nicknamed RADD for Requirements Analysis tool with Data Dictionary) that allows a user to interactively create and edit structured analysis diagrams while extracting data dictionary information from the diagrams (Urscheler, 1986:3). By integrating these techniques into one tool, the software analyst's work is reduced when creating data dictionaries and structured analysis diagrams.

In conjunction with this effort, another thesis was conducted to design a central database that will standardize the data formats for software engineering tools (Connally, 1987). Thus, a criterion for this thesis was to develop a data format capable of interfacing to this central database as well as storing complete data dictionary information and structured analysis graphics information.

Background

SADT. SADT is the name of SofTech's methodology for doing requirement analysis and system design. It was first published in 1977 by Douglas T. Ross of SofTech (Ross, 1977). The methodology was found effective when applied to "a wide range of planning, analysis, and design problems

involving men, machines, software, hardware, database, communications procedures and finances ..." (Ross, 1977:17).

SADT provides techniques and methods for:

1. thinking in a structured way about large and complex problems;
2. communicating analysis and design results in clear, precise notation;
3. controlling accuracy, completeness, and quality by procedures for review and approval;
4. documenting the system analysis and design history, decisions, and current results;
5. working as a team with effective division and coordination of effort; and
6. managing development projects and assessing progress (Softech Inc., 1977:1-1).

An SADT model consists of diagrams drawn according to a well defined graphical syntax. Each diagram has accompanying text to assist in the understanding of the diagram. The diagrams represent a hierarchical outline of the system. According to Softech

Each lower-level diagram shows a limited amount of detail about a well-bounded subject. Further, each diagram connects exactly into the model to represent the whole system, thus preserving the logical relationship of each component to the total system (Softech Inc., 1976:2-6).

Figure 1 is an example of an AFIT SADT type diagram. SADT models all systems in terms of the system happenings, or activities, and the system objects, or data (Softech Inc., 1976:2-5). Softech SADT methodology calls for the

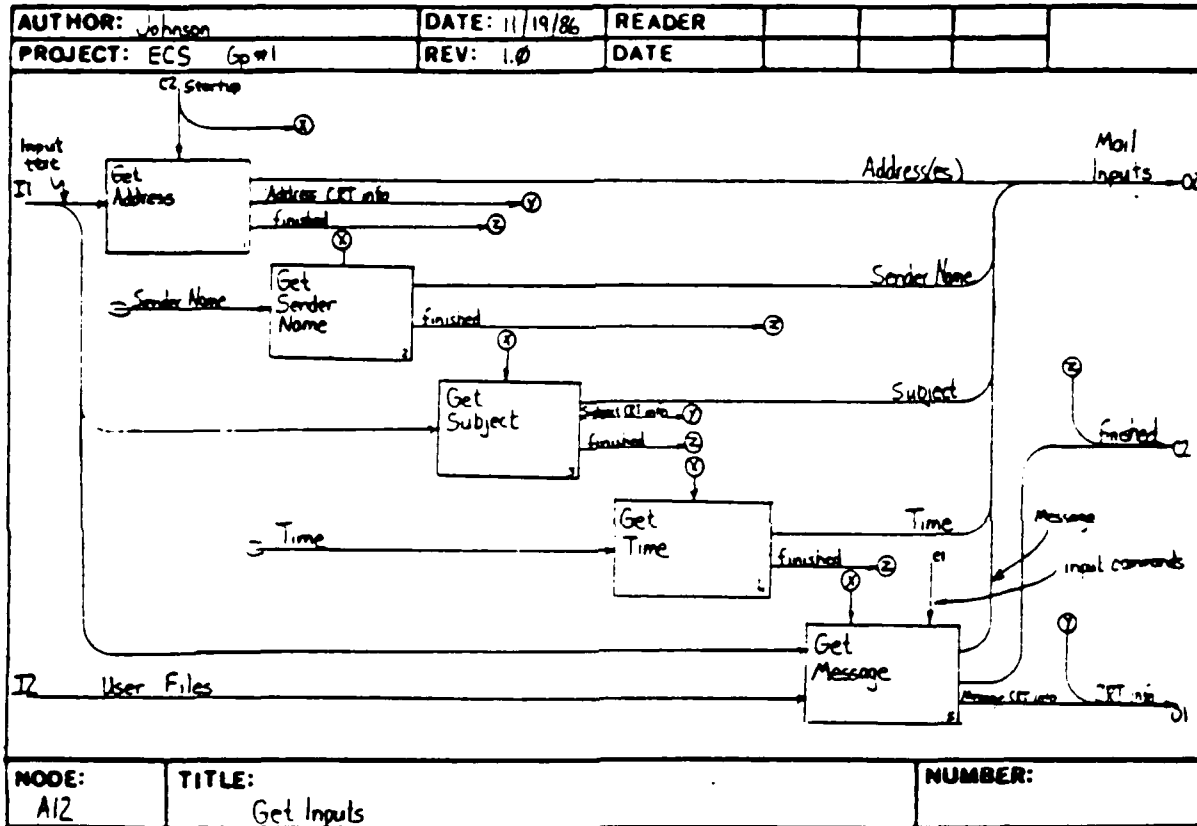


Figure 1. Example SADT Diagram

decomposing of both the activities and the data independently of each other (Ross, 1977:19).

The language used to construct both types of SADT diagrams consists of 40 graphical type features and principles. The language has two components: English text and graphical constructs. Combining the two components provides the function needed to provide communication (Ross, 1977:19).

The graphical language is designed to

1. expose detail gradually and in a controlled manner;
2. encourage conciseness and accuracy;
3. focus attention on module interfaces; and
4. provide a powerful analysis and design vocabulary (Softech Inc., 1976:2-6).

Rectangular boxes and arrows are the primary graphical constructs used in making an SADT diagram. The boxes represent the decompositions of the parts of the system. Arrows are used to describe how the boxes interface between each other on the diagram. English text is used to label the boxes and the arrows and define their meaning (Softech Inc., 1976:4-4). According to the Softech definition

Input data (on the left of the box) are transformed into output data (on the right) by the activity represented by the box. Controls (on the top) govern the way the transformation is done (Softech Inc., 1976:4-5).

Arrows do not represent a flow of control, rather, they describe classes of data. One arrow can represent several

classes of data; this is referred to as a pipeline. The label for the arrow must precisely describe the contents of the pipeline (Softech Inc., 1976:4-7).

Applying the SADT methodology allows teams to effectively work together. As part of the process, the authors distribute drafts of the SADT diagrams and supporting documentation to interested parties. This process, called a peer review, permits the parties to review the drafts and make written comments to the authors. The author then responds to these comments. This process continues until all agree that the model is complete.

The reader is referred to the 1977 article by Douglas T. Ross for a more definitive description of the SADT concept (Ross, 1977).

IDEF₀. The U. S. Air Force Program for Integrated Computer Aided Manufacturing (ICAM) adopted a structured graphic technique called IDEF₀ (ICAM Definition Method zero). IDEF₀ is a derivative of SADT, tailored by Softech, Inc. for ICAM. It is the function modeling technique applied to all ICAM projects (Smith, 1981:1). IDEF₀ was developed to give a structured approach to applying computer technology to manufacturing and to enhance the communication and analysis of the people involved in improving manufacturing productivity. A detailed comparison of the SADT and IDEF₀ syntaxes is found in Appendix A.

Data Dictionaries. The purpose of a data dictionary is to manage and document a valuable resource, data. Using a data dictionary system properly provides measurable benefits to an organization (Lefkovits, 1977). Ten areas that benefit from an effective data dictionary system are listed below.

1. Reduction of data redundancy
2. Reduction of system development costs
3. Enhancement of the system maintainability
4. Improved impact of change assessments
5. Enforcement of data standards
6. Improved information for data base creation
7. Improved Communication between people
8. Better auditing of use of data
9. Reduction of Administrative Effort
10. Creation of trustworthy information (Lefkovits, 1977:1-8 thru 1-11)

The AFIT methodology for including data dictionaries with the requirements analysis is supported by Leong-Hong and Plagman. They said,

The use of the DD/DS (data dictionary/directory system) in requirements definition and analysis is critical. The DD/DS provides a framework in which the end-user and the analyst can communicate with each other using common terminology and definitions. Communication between the end-user and the analysts, between the analyst and the designer, and between the designer and the developer is essential in building a system. By maintaining consistency in the data used, potentially disastrous conditions caused by inexact or inconsistent data can be averted (Leong-Hong and Plagman, 1982:34).

Leong-Hong and Plagman identified yet another benefit that the data dictionary gives the requirements analyst, that of maintaining documentation. The data dictionary maintains descriptions of the activities and data needed for each

activity. From these descriptions, documentation regarding the effect of activities upon each other is readily available (Leong-Hong and Plagman, 1982:41-43). The authors further recommend an automated data dictionary system for producing documentation to reduce the monotony and repetitiveness of the task (Leong-Hong and Plagman, 1982:50).

Statement of the Problem

The purpose of this thesis was to integrate the structured analysis and data dictionary techniques into a CAD tool to attempt to make the software requirements analyst's job more efficient.

Scope

To make RADD useful, the requirements and design of the RADD prototype were analyzed to ascertain the changes and additions needed. The software to effect the changes was designed, coded, and tested, and the usefulness of the new version of RADD was established by a formal evaluation using questionnaires and statistical analysis of the responses.

Assumptions

For the purposes of this thesis, the researcher made four assumptions. These assumptions are listed below.

1. The users of this tool are AFIT graduate students and faculty.
2. The users of this tool are competent in the use of computers.
3. The users of this tool are familiar with the structured analysis methodology and uses of data dictionaries.
4. The requirements analysis methodology used is described in AFIT's Software Development Documentation Guidelines and Standards (Hartrum, 1986:8).

Research Approach

This research was accomplished in three phases. In the first phase, the current requirements analysis for RADD was reviewed, followed by a review of the design and implementation of RADD. These were updated and changed as necessary to reflect the new requirements and design of the new tool. Next, the reusable parts of the RADD software were extracted and new code written to implement the changes deemed necessary by the analyses. Finally, an evaluation of the new tool's usefulness to a software analyst was accomplished.

Part of the first phase was to review the requirements analysis for RADD. This was necessary because the initial version of RADD needed improvements to become useful; therefore, requirements for the initial version of RADD were not complete. The requirements analysis for RADD was described textually in Chapter Three and graphically by structured analysis diagrams in the Appendix of the

Urscheler thesis. In performing this analysis, it was necessary to analyze other structured analysis methodologies to define a suitable language for the AFIT environment. From this language, a minimum subset for the new structured analysis tool was identified.

Also during the first phase, the design and implementation of RADD was analyzed. This was necessary because RADD was not completely useful. For instance, in addition to expanding the language implemented by RADD, it was necessary to permit printing hard copies of the structured analysis diagrams. Furthermore, design issues such as extensibility and maintainability were considered. The design of the database was examined with regards to the central database management system being designed for the AFIT distributed design environment. After completing these analyses and identifying the necessary enhancements, the second phase began.

The second phase was the development of the software. Using a top-down approach, the modules were coded and tested, integrating the system in the process. All software conformed to the standards set forth in AFIT's Software Development Documentation Guidelines and Standards pamphlet (Hartum, 1986).

The third phase was a formal evaluation of the structured analysis tool's usefulness to a software systems analyst. A likely pool of analysts, familiar with

structured analysis and data dictionaries, was drawn from the students and faculty of AFIT's software engineering classes. These people were polled using standard questionnaires (Foley, 1986). The results of these questionnaires were compiled and analyzed with statistical methods, establishing the tool's usefulness to the AFIT software systems analyst.

Equipment and Software

The equipment and software for this thesis were available in the Information Systems Laboratory of the AFIT Department of Electrical and Computer Engineering. The computer used for this thesis is the Sun 3 (Sun is a trademark of Sun Microsystems Inc.) workstation. This workstation runs Berkeley UNIX (UNIX is a trademark of AT&T) version 4.2 and features Suncore graphics and the Sunwindow environment. The software developed in this thesis was written in C.

Sequence of Presentation

This thesis consists of six chapters. A literature review of human/computer interface issues is presented in Chapter II. The requirements for the new tool are presented in Chapter III. The system design is presented in Chapter IV. Chapter V is a summary of the implementation and is a statistical analysis of user reaction to the tool. Chapter VI presents the researcher's conclusions and recommendations.

II. Review of the Literature

Introduction

The purpose of this thesis effort was to build an interactive CAD tool. It was imperative that the researcher understand the issues that impact interactions between the computer and the user. Since the human/computer interface issues were deemed critical to the success of this effort, a review of the literature was conducted to gain the knowledge needed to design the human/computer interface for this tool.

Human/Computer Interface

A computer system's effectiveness is directly related to how well the user and the computer are able to communicate with each other. Newman and Sproull noted it is the design of this user interface that "... has a particularly strong impact on the program acceptability as a whole" (Newman and Sproull, 1979:443). The importance of the human/computer interface was further amplified by Robert W. Bailey. He said:

Not considering human performance in the human/computer interface frequently results in large numbers of errors, requires huge amounts of training time, and causes user frustration and dissatisfaction (Bailey, 1982:293).

Because each user has a different background and experience, the process of designing the user interface is difficult. This is further complicated because a

universally accepted method for designing a human/computer interface does not exist (Woffinden, 1984:15-16). As Foley and Van Dam noted

Like architecture, the design of user interfaces is at least partly an art rather than a science. We hope that the design of user interfaces will someday become more science than art, but the climb to reach this goal is long; the ascent has begun, but there are many hard traverses ahead (Foley and Van Dam, 1982:217).

User Interface Components. Newman and Sproull (Newman and Sproull, 1979:445) divided the user interface into four components. Figure 2 enumerates these components. Since the names of the components do not imply their full meaning, a short discussion of each follows.

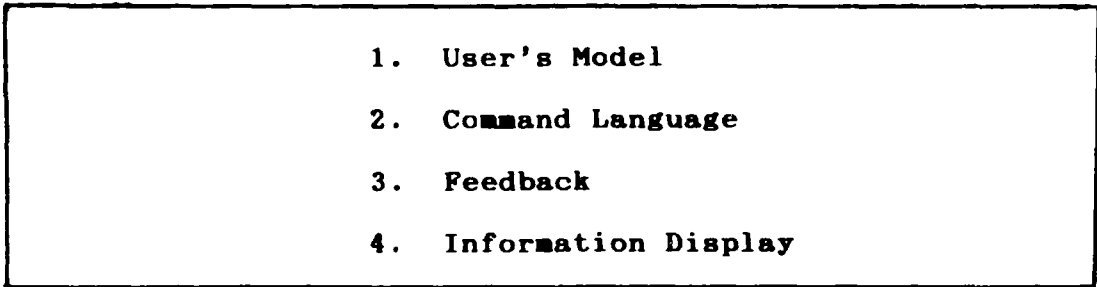
- 
1. User's Model
 2. Command Language
 3. Feedback
 4. Information Display

Figure 2. Components of the User Interface

The user's model is "the conceptual model formed by the user of the information he manipulates and of the processes he applies to this information" (Newman and Sproull, 1979:445). In other words, it is the way the user thinks and understands the program to work; therefore, he is able to devise his own strategies for operating the program.

Thus a good user model is characterized by the user understanding the purpose of each input he gives the computer rather than blindly following the user's manual instructions. The user's model should present objects or phrases that are familiar to the user and simulate the modeled environment. "The use of familiar concepts makes the user's model more intuitive and easier to learn" (Newman and Sproull, 1979:448).

Next, the user needs to know the command language to use the model. Designing a command language involves recognizing that all commands relate to each other and together they define a syntax for the language. In addition to syntax, the semantic value of each command must be considered.

Newman and Sproull reported there are four issues that further complicate the design of a command language. These are

1. Command modes. Allowing the same user input to be interpreted in different ways depending on the current mode.
2. Selection sequence. Usually an operation must be selected in addition to selecting an object to operate on. Which comes first has an effect on the number of command modes.
3. Command abort mechanism. Commands requiring a sequence of inputs must allow for retraction of the command in the middle of the sequence.
4. Error handling. It must be decided how to handle erroneous or meaningless commands (Newman and Sproull, 1979:451-452).

Another dimension to the user interface design complexity is the number of possible input devices available. The designer must be consistent by making the user operate the same input device for each command.

Feedback is required to let the user know what is going on in the program, and should be given quickly. According to Newman and Sproull, three important forms of feedback are

1. Feedback by informing the user if a command has been accepted, or if an error has occurred.
2. Feedback that the correct object has been selected.
3. Feedback such as cursor feedback, character echoing, and so forth (Newman and Sproull, 1979:464).

Information display is the final component of the user interface. The flexibility given designers by graphical devices poses a problem in determining an effective means to display the information. These problems have fallen into two categories: overall layout and representation of objects. Screen space is usually at a premium and what is pertinent for display at a given time must always be evaluated. On the other hand, if user controlled windows compose the display, the burden of arranging the display can be shifted to the user (Newman and Sproull, 1979:460).

Newman and Sproull further suggested that the quality of the graphic display forces the designer to make trade-offs. The designer must decide between the amount of screen

space taken and the amount of detail necessary for a given image (Newman and Sproull, 1979:462).

After studying the components of the human/computer interface, various general user interface design principles were researched.

Design Principles. As previously stated, a universally acceptable method for designing a human/computer interface does not exist; however, experience has led to the documentation of certain design principles. Figure 3 is a list of principles by four different authors for designers of interactive programs. It should be noted the principles encompass all four components of the user interface. Also, the list given by D. S. Woffinden was derived from works including those of the other three authors on the list.

Woffinden found these guidelines incomplete. He said:

None of the lists of general design guidelines studied were found to be completely satisfactory. Many seem to have become over concerned with the human issue and seem to fall short of giving guidance revelant [sic] to the complete design process for an interactive system (Woffinden, 1984:19).

Figure 3 shows that each author views human/computer design from a slightly different perspective. Nonetheless, Foley and Van Dam asserted it is important for the designer to consider these design principles if a satisfactory interface is to result (Foley and Van Dam, 1982:218).

Bailey ^A	Foley and Van Dam ^B	Hansen ^C	Woffinden ^D
Understand the User, Activity and Context	Give Feedback	Know the User	Determine purpose of the system
Ensure User Control	Help the User Learn the System	Minimize Memorization -Selection not Entry -Names not Numbers -Predictable Behavior -Access to System Information	Know the user
Maintain Consistency	Allow Backup and Accommodate Errors		Identify resources available
Minimize Processing Time by People	Control Response Time	Optimize Operations -Rapid execution of common operations -Display inertia -Muscle Memory -Reorganize command parameters	Consider human factors
Keep the User Informed	Design for Consistency		Determine the interface language
Improve User Accuracy and Facilitating Error Handling	Structure the Display	Engineer for Errors -Good error messages -Engineer out common errors -Reversible actions -Redundancy -Data structure integrity	Consider the environment of the operation
Optimize Training	Minimize Memorization		Design for evolution
			Optimize training
			Accommodate levels of experience
			Use selection vs. entry
			Be consistent
			Anticipate errors

Figure 3. Design Principles
 (Adapted from: ^ABailey, 1982: 297-299;
^BFoley and Van Dam 1982:222-238;
^CHansen, 1971: 24-228; ^DWoffinden, 1984: 20)

Interfacing with a Mouse

Increasing the number of different input devices increases the design complexity. Since, in this project, the mouse and the keyboard were used for input, it was important to consider the issues surrounding human/computer interfacing with a mouse.

Lynne A. Price considered it practical to use a mouse in CAD programs because of the following six advantages:

1. Mice allow control of the cursor through arm motion natural to pointing.
2. A mouse gives the user the ability to point and to invoke several functions with one hand.
3. The mouse allows cursor position to be controlled by software somewhat independently of the user's hand motion.
4. The mouse permits the user to remove his hand from the pointing device (e.g., to use the keyboard or answer the telephone) with no change in the cursor position on the screen.
5. The pads required by optical mice are light, are easily moved, and may be placed on top of normal desktop clutter.
6. Mice are convenient for both left and right handed users (Price, 1984:288).

Price conducted four experiments to test the suitability of the mouse as a pointing device for CAD systems. The experiments involved pointing the cursor with the mouse at graphic items and clicking the buttons. A mechanical, three button mouse was used for the first three experiments (Price, 1984:288-293).

The first experiment tested forty-two individuals' abilities to use the buttons and control the cursor position. Errors occurred more frequently when the number of buttons involved increased and when the mouse was moved while buttons were held down (Price, 1984:289).

The second experiment attempted to determine if different numbers of clicks of a single button or use of a different button to indicate the same input decreased accuracy and caused more errors. No significant difference was observed in the time to complete comparable experiments or in the accuracy between the two methods (Price, 1984:290).

The third experiment attempted to determine if the decrease in performance noted in the previous experiment was a function of the time allowed the user to begin the second click. In experiment two, the people were required to enter the second click within a half to three-quarters second of the first click. In this experiment, this delay was varied up to one half, three-quarters, and one full second. Price found as the delay time was increased, the difference in performance became less significant (Price, 1984:291).

In the final experiment, a different style of mouse was tested, one with two horizontal rocker buttons providing four distinct inputs. No significant difference in performance or accuracy was identified for any for the variations tested; however, a large majority preferred the

variation that most closely simulated the previous style mouse. Price concluded that people prefer using one finger for each different input and that a mouse with vertically configured buttons is preferred (Price, 1984:292).

Overall, Price concluded the differences in performance and accuracy identified in the four experiments were small enough to justify using more complicated input techniques when the number of different inputs exceeded the number of available buttons. She also noted these more complicated techniques would cause a large software overhead for the designer by requiring a check for multiple button clicks and by requiring a check for user input errors (Price, 1984:293).

Summary

The purpose of this literature review was to assimilate the latest information that pertained to the interface between a computer and its user. Information was gathered to review the components of the interface and the design principles one should consider when constructing a human/computer interface. Also, because the tool constructed by this thesis effort used a mouse as an input source, information on experiments to test a mouse's suitability for the tool was studied. After gaining an understanding of these critical subjects, the researcher based requirement and design decisions on the foundations learned here.

III. REQUIREMENT ANALYSIS FOR A STRUCTURED ANALYSIS TOOL

The issues constraining the requirements for this tool are sorted into three broad categories. The categories are available hardware support, available software support, and existing requirements based on previous studies. After considering these constraints, the requirements specific to the CAD tool are identified.

Hardware Support

Because this tool is to be integrated into the AFIT computing environment, it is necessary to consider the restrictions the environment poses on the tool. Currently, the computing environment consists of several mainframe computers and many stand-alone workstations. The primary mainframes are two VAX 11/785 computers (VAX is a trademark of Digital Equipment Corporation Inc.) running the Berkeley 4.3 UNIX operating system.

Access to the mainframes is available through the use of the AFITNET. This provides the capability to connect to the computers with a home computer over the telephone lines (AFIT/SI, 1987). Also, there are Zenith Z-100 and Z-248 workstations that are able to access the AFITNET. The AFITNET gives other mainframes and the SUN (SUN is a

trademark of SUN Microsystems) workstations the capability to remotely login to the 11/785's via an Ethernet cable.

Software Support

As discussed further in the next section, a requirement already existed that the data produced by the tool be stored in a relational database using the INGRES database management system on a UNIX host computer. The capability to store the data produced by the tool was developed in conjunction with this thesis effort. Thus, there was a requirement that the software create data files usable by the relational database manager.

The software used to create the tool was also required to interface to a graphics package. In the interests of portability, the use of a standard graphics package needed consideration. Also, the software needed to be portable to other systems.

Existing Requirements from Previous Studies

Data Dictionary Editor. This computer tool permits users to create and edit data dictionary definitions for the requirements, design, and coding phases of the software lifecycle. After editing the definitions on a personal computer type workstation, users may transfer the definitions to the database host via the AFITNET using available communications software. A current thesis

effort will make it possible to store the definitions in the central database.

The data dictionary definitions created by the editor and the tool from this thesis are related. Urscheler designed the prototype in this manner. He said:

To insure there is a level of consistency, the same definitions were employed in RADD. The consistency resulted in a direct link between the Z-100 editor and RADD at the database level (Urscheler, 1986:19).

Thus the requirement to maintain consistency at the database level still existed for this thesis effort.

Human/Computer Interface Requirements. As discussed in Chapter Two, an adequate human/computer interface is critical to the success of interactive computer tools. This thesis effort as well as the Urscheler effort were bound by this requirement. Figure 4 is a list of the five key psychological factors Urscheler considered in attempting to fulfill the human/computer interface for his effort.

Analysis of the RADD human/computer interface by this researcher found the interface a good one. Overall, the user feels a real sense of drawing an SA (Structured Analysis) diagram when using RADD. However, enhancements in areas specified by the requirements in Figure 4 are needed to improve the existing human/computer interface. The requirements presented by Urscheler and those established in Chapter Two were considered in the design of the follow-on tool.

1. Keep the user motivated -- do not frustrate or bore him.
2. Break the lengthy input process into parts to permit the user to achieve "psychological closure". This provides positive feedback to the user through a feeling of accomplishment and success.
3. Minimize the memorization required by the user.
4. Provide visually pleasing displays on the screen. This includes minimizing the scrolling and other distracting movements of text, the highlighting of instructions to the user, and making effective use of margins and white space.
5. Keep response time to a minimum. Display status messages to keep the user constantly informed of what is happening inside the machine.

Figure 4. RADD Human-Interface Requirements
Source: (Urscheler, 1986:21)

Requirements for the Follow-on Tool

In defining the requirements for the follow-on tool, it was necessary to analyze the current implementation and determine how well it conformed to the original requirements. Areas of specific interest were the human/computer interface, the tool's functionality, and the reusability of the software. Also, it was necessary to evaluate the SADT methodology to determine the requirements for a necessary and sufficient SA language suitable for the AFIT software development environment.

Determining how well RADD complied with its requirements was difficult. Being a prototype and given the time constraint in which it was developed, the requirements analysis did not measure up to those specified in the Software Development Documentation Guidelines and Standards. The given requirements analysis was not decomposed to the level of detail necessary to specify all the activities and data.

RADD implemented a small subset of the total SADT language needed to satisfy the AFIT software requirements methodology; therefore, a requirement existed to define an exact language syntax for the tool. If possible, it was desirable to develop a standard consistent with existing standards adopted by the Air Force. A study was conducted to determine if such a standard existed and to define the syntax. Appendix A discusses the results of the study.

In addition to the SADT diagram, requirements existed to produce facing page text for the diagram and data dictionaries for each activity and data element. Format guidelines for these products are given in Software Development Documentation Guidelines and Standards. Appendix A also presents exact formats for each of these products.

Summary

This chapter described the requirements for a graphical structured analysis CAD tool. The requirements based on constraints from hardware support, software support, and previous studies were discussed. Also, human/computer interface requirements were identified and discussed. The next chapter presents the design decisions based on these requirements.

IV. DESIGN AND DEVELOPMENT OF THE SA TOOL

In the previous chapter, specific requirements for this tool were discussed as well as issues constraining the requirements. The purpose of this chapter is to present the design decisions made based on those requirements and on the objectives of the thesis. This chapter discusses the design considerations taken into account during the design process, the considerations in defining the internal data structures used, the definition of the file structures used, and the approach taken to the coding and testing of the tool.

Hardware Considerations

The SUN workstations were chosen because they met several hardware requirements. First, the SUN workstations are linked to the AFITNET. This integration with the AFIT computing environment was important because data dictionary information from the tool is stored in a database that may be maintained on another machine.

The SUN workstation's main memory is adequate for executing the tool's program. Because the executable file is in excess of 1 Mbyte, smaller workstations like the Z-100 could not be used.

The SUN workstations have a large display monitor that accommodated the intended screen layout. The intended screen layout consisted of five distinct areas, one being

the drawing area that had to be large enough to clearly show all the graphical constructs of the SA syntax.

The SUN workstations provide a mouse in addition to the keyboard as an input device. The mouse is the primary input tool for selecting and manipulating the graphical constructs on the SA diagram because of its ability to function as both a locator and a pick device.

Finally, the SUN workstations are sufficient in number and meet the availability requirement. Currently, there are six SUN workstations available and all are linked to the AFITNET.

Software Considerations

Portability was considered when selecting a software graphics package. The prototype developed by Urscheler utilized the SunWindows environment (SunWindows is a trademark of Sun Microsystems Inc.). At the time the tool implementation began, an implementation of the GKS graphics standards called SunGKS was available only in a beta version with the release date of the final version unknown. Also at implementation time, SUN had released a new version of SunWindows, called SunView. Given these circumstances and that portions of Urscheler's software could be used, it was decided to proceed using SunView, attempting to isolate each Sunview function call within its own module as much as possible. Also, using an available graphics package ensured the successful implementation of a complete tool.

This decision limited the language used to C. This choice did not, however, detract any more from the portability aspect of the tool.

Previous Study Considerations

Since data dictionary information may be entered into the data dictionary database from sources other than this tool (currently the Data Dictionary Editor is the only other tool with this capability), there were existing constraints on the field lengths for the data dictionary entries. To meet this requirement, the field lengths specified in the Data Dictionary Editor were used in this tool.

From Urscheler's thesis effort, two design decisions were carried over. First, the window size was kept the same because it closely represents an eight by eleven inch page size. Second, the size of the activity boxes was kept constant. Urscheler's testing found that user-determined box sizes led to increased software and data structure complexity (Urscheler, 1986,27).

Human/Computer Interface Considerations

The importance of an acceptable human/computer interface was discussed in Chapter Two and Chapter Three. To attain this objective, the design decisions surrounding the screen layout, the menus, and the use of voice for feedback are presented in the following paragraphs.

Screen Layout. Figure 5 is a picture of the actual screen layout used by the tool. It was decided to divide the screen into five distinct areas or windows: the Input Window, the Message Window, the Selection Window, the Diagram Window, and the Data Dictionary Window. Each of

SA TOOL TOOL					
INPUT: DISABLED					
MESSAGE: WELCOME. Please make a selection.					
RECALL DIAGRAM		EDIT DIAGRAM		EDIT DD	
		EDIT FPI		MISC FUNCTIONS	
SAVE DIAGRAM					
AUTHOR:		DATE:		READER	
PROJECT:		REV:		DATE	
NODE:	TITLE:				NUMBER:

Figure 5. SA Tool Screen Layout

these windows, except the Data Dictionary Window, serves a unique purpose; the Data Dictionary Window serves two purposes. The following briefly discussed the role of each window.

Diagram text is typed from the keyboard and is echoed in the Input Window, which is located at the top of the screen layout. For each input, a maximum length is defined. Attempts to add to the input beyond this limit are not allowed. Also, errors are correctable by using the "DELETE" key to backspace over errors. After typing the input, striking the "RETURN" key puts the text on the diagram.

The Message Window, located directly beneath the Input Window, is integral to the user interface because it is the primary means by which the tool keeps the user abreast of the tool's status. Here the user receives instructions on how to proceed with a given operation, feedback on the results of a particular action, and help messages on how to resolve a problem that occurred. The words "Make another selection" are used to indicate that the current operation is completed and the program is ready to process another menu selection. The menu system is discussed in more detail in the next section of this chapter.

The Selection Window, located directly below the Message Window, is where the user selects the menu that contains the next desired operation. Five ovals are laid

out in the left-to-right order in which it is anticipated the user would pick the menus. For example, it is anticipated the user would first edit the diagram by adding and deleting lines, boxes and labels. Next, the user would enter the applicable data dictionary information before getting hardcopy outputs of the data dictionaries, the facing page text, and the diagram. Finally, the user would save the diagram for possible future editing.

The Diagram Window is located underneath the Selection Window and is where the SA diagram is actually "drawn." In the Diagram Window the user is able to create, delete, and move activity boxes, lines, footnotes, and squiggle lines. All of the menu selections for manipulating these and other graphical entities are accessed by selecting the "EDIT DIAGRAM" oval in the Selection Window.

Finally, the Data Dictionary Window is a dual-purpose window located below the Diagram Window where data dictionary information that cannot be derived from the graphical inputs is entered. The user enters the needed information from both the keyboard and the mouse. Using the mouse, the user quickly moves the cursor to the particular field in which he desires to enter the text. The Data Dictionary Window also displays the current facing page text.

Menu System. Considering the nature of the tasks to be accomplished, a menu driven system was chosen because it

satisfied several of the rules for a good user interface. Specifically, a menu driven system offered the following advantages:

1. It automatically partitioned the input process each time the user selects another operation.
2. It minimized the memorization required by the user.
3. It gave the user a feeling of control over the tool by selecting the next operation.
4. It reduced the number of opportunities for the user to make an error by having to type the selection.
5. It permitted a uniform selection process.

The user chooses a menu of operations from among the Selection Window ovals. Each oval, except the "RECALL DIAGRAM" selection, puts a menu of operations on the screen from which the user selects.

In addition to the left-to-right ordering of the ovals, each menu was designed in a hierarchical manner that matched the functional decomposition of the tool. Figure 6 shows all the menu selections used by the tool and their decompositions. The pictures are the actual screen images seen by a user of the tool, with the arrows on the right of the menu meaning there are more selections to be chosen. For a detailed explanation of each of the menu functions, the reader is referred to the User's Reference Manual, Appendix I of this document.

RECALL DIAGRAM

EDIT DIAGRAM

EDIT DD

FPT FUNCTIONS

MISC FUNCTIONS

SAVE DIAGRAM

EDIT Line
EDIT Activity Box
EDIT Header Info
DELETE Footnote
DELETE Squiggle
ADD Activity Box
ADD Header Info
ADD Line
ADD/CHANGE Footnote
ADD Squiggle

ADD DD
ADD MORE ALIASES
EDIT DD
SAVE DD IN FILE
DD-FINISHED

DISPLAY FPT
SAVE FPT IN FILE

Make Diagram (Normal)
Make Diagram (Sideways)
Change Directory
Display Directory
START New Diagram
Redisplay Diagram
QUIT (NO SAVE)

Save for DB
Save Local

Edit Line Label
MOVE Line Label
Edit TO/FROM Label
Edit ICON Label
Redraw/Delete Line

Change Activity Name
Change Activity Number
Change Activity Box Location
Delete Activity Box

Edit Author
Edit Project
Edit Date
Edit Revision
Edit Node
Edit Title
Edit Number

Add Author
Add Project
Add Date
Add Revision
Add Node
Add Title
Add Number
Add ALL

Boundary Arrow
Tunnel Arrow
From ALL
Dot-T/R
Dot-B/L
Arrowhead
DONE

Boundary Arrow
Tunnel Arrow
To ALL
Dot-R
Dot-L
Arrowhead
Turn-R
Turn-L
Branch-R
Join-R
Join-L
DONE

ADD Footnote
Change Footnote Number
Change Footnote Location

Figure 6. SA Tool Menus

Making a selection from any of the menus is a uniform process. With the mouse, the user places the cursor over the text of the desired menu selection. When the selection is changed to reverse-video, the selection can be made by clicking the left button on the mouse.

Voice Feedback. This tool was designed to accommodate a DECTALK (DECTALK is a trademark of Digital Equipment Corp.) voice synthesizer running version 2.0. The objective of this decision is to enhance the human/computer interface by attempting to permit the user to concentrate his attention more on the Diagram Window than the Message Window. To activate this option, SATool must be executed with a -v option to initialize the DECTALK. A software module called "put message" directs messages to either the DECTALK, the Message Window, or both. Due to the time constraints, this capability was installed in the tool, but not developed or evaluated.

These paragraphs discussed the screen layout, menu setup, and voice feedback issues in the design of the tool. In the next section, the data structure design issues are discussed.

Data Structures

The internal data structures developed for this project were entirely different from those developed in the Urscheler prototype (Urscheler, 1986:A-1). Urscheler's data structures were designed only for the small subset of the SA

syntax used by his tool; therefore, new data structures were needed to implement graphical entities like footnotes and squiggle lines. In addition, this researcher's analysis of the prototype concluded that the low-level approach of treating all lines (whether they made up an activity box or lines connecting the boxes) and all text (whether it labels the diagram, a box, or a line) as the same entity resulted in increased software complexity. Based on this analysis, a higher level, object oriented, approach was taken.

This section is divided into two parts. First, the primary data structures and the information they store are described. Second, the data dictionary information derived from the diagram is discussed.

Primary Data Structures. The design of the data structures was driven largely by the requirement for the tool to produce one separate file of data dictionary information that is capable of being stored in a relational database. This requirement led to the following design objectives:

1. The data structures must maintain both graphics and data dictionary information.
2. The data structures must separate the data dictionary information from the graphics information as much as possible.
3. The data structures must maintain enough information to allow the graphics and data dictionary information to be stored in separate files and later restored from those files.

From these objectives, five primary data structures were designed to hold all the graphics and data dictionary information. The following paragraphs describe each of these structures and how the data dictionary information relates to the structure.

The box structure contains the information needed to locate and label activity boxes as well as store data dictionary information not derived from the diagram. In addition, each box structure is classified as such by a numeric structure type field. All the activity boxes on the diagram are maintained by using a linked list; therefore, a C pointer to the next box structure was also defined.

The box structure uses another C pointer to point to an activity data dictionary structure. This structure saves the data dictionary information input by the user and is merged with the diagram information to complete the data dictionaries for activities. Specifically, the user must input the description field, alias name field, alias comment field, version changes comment field, and related requirement number field.

The line structure contains the information needed to locate, label, and draw the lines as well as store the data dictionary information not derived from the diagram. Each line is given a numeric structure type that identifies it and specifies how it connects to other lines. In addition to the label identifying the data it represents, a line may

have ICOM labels attached to it, specifying the line as a boundary arrow, and a label that is placed inside a TO-ALL or FROM-ALL circle attached to one end of the line. Also, two numbers are defined that identify the graphical entities drawn on each end of the line (ie. arrowhead, tunnel, dot, turn right, or branch left, etc.). Finally, the lines are stored in binary trees with the root nodes linked to other root nodes by C pointers. Figure 7 shows three groups of lines and the corresponding linked list structure is shown in Figure 8. It can be seen from this figure that this arrangement is advantageous because the tree arrangement maps closely to how the line segments actually connect to one another and because C supports the simple recursive functions used for traversing binary trees.

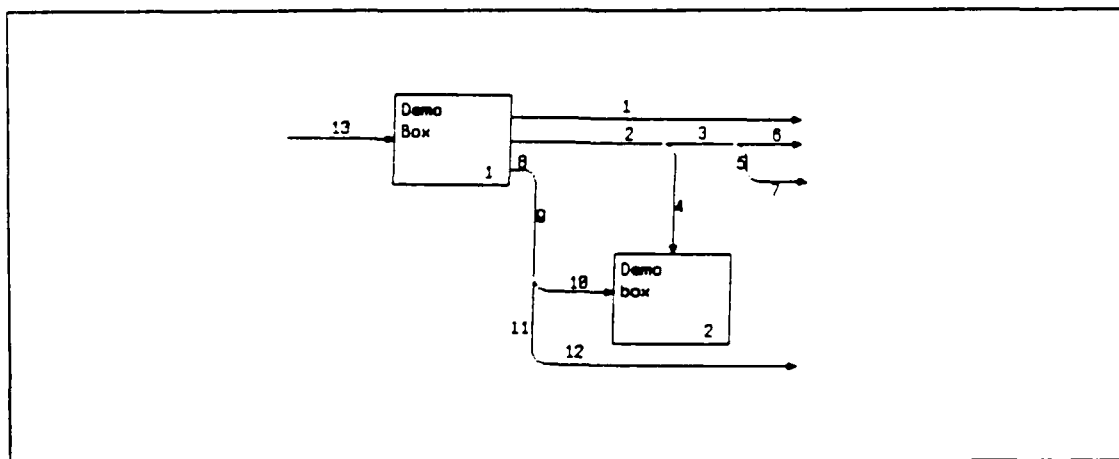


Figure 7. Example Group of Lines

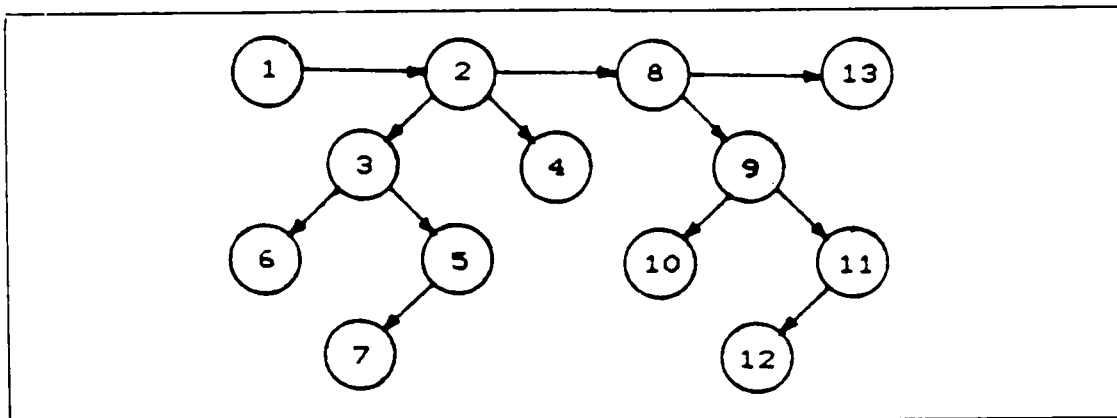


Figure 8. Resulting Linked List Structure

The line structure uses a C pointer to point to a data dictionary structure representing a data dictionary for a data element. This structure saves the data dictionary information input by the user and is merged with the diagram information to complete the data dictionaries for data elements. Specifically, the user must input the description field, alias name field, alias comment field, related requirement number field, version changes comment field, data type field, minimum value field, maximum value field, range of values field, values field, part of field, and the composition field.

The squiggle line structure contains all the information needed to draw a squiggle line. A squiggle line is strictly a graphical entity, needing only four coordinate pairs to complete its specification. Again, a squiggle line classification number is assigned to each squiggle on the diagram. The squiggle lines for a particular diagram are

stored in a singly linked list; therefore, each structure contains a C pointer to another squiggle line structure.

The header structure contains all the information needed to draw, locate, and classify the seven header fields of an SA diagram. Except for the number field, each of the fields is included in the data dictionaries for the diagram. Since each diagram only has one header, only a single C pointer is maintained to save this information.

The last primary storage structure, the footnote structure, keeps all the information needed to draw, locate, and classify a matching pair of footnote labels. Like the squiggle line, this information is strictly graphical. The footnote structures for a diagram are stored in a singly linked list; therefore, a C pointer to another footnote structure is defined.

Data Dictionary Information Derived from the Graphics Information. So far in this section, the graphical information stored in the various structures and the data dictionary information not contained in the graphics information has been identified. The purpose of the following paragraphs is to identify the data dictionary information that is derived from the graphics information.

In an activity data dictionary, the header structure directly provides five data dictionary inputs and part of a sixth. The PROJECT, DATE, and AUTHOR fields are exact matches with the fields of the same title in the data

dictionary. The NUMBER field of the data dictionary is the result of appending the number of the activity box to the NODE field of the diagram. The PARENT ACTIVITY and VERSION fields of the data dictionary are synonymous with the TITLE and REV fields of the diagram, respectively.

The INPUTS, OUTPUTS, CONTROLS, and MECHANISM fields of the data dictionary are derived by matching the starting and ending points of a line segment with the boundary of the activity boxes. When one of the line's start or end points matches a box boundary, the side of the activity box determines the appropriate field and then tree traversal algorithms can find the closest label to the box.

In a data dictionary for a data element, the header structure directly provides four data dictionary inputs. The PROJECT, DATE, and AUTHOR fields are exact matches of fields with the same title in the data dictionary. Also, the VERSION field of the data dictionary is the same as the REV field of the diagram.

The SOURCES and DESTINATIONS field of the data dictionary cannot consistently be resolved by the tool. It is possible in an SA diagram to have a valid data entity that has no sources or destinations due to the pipeline feature of the language. Currently, the data dictionary is unable to handle this situation with the precision needed for implementation with the tool; therefore, the tool leaves these spaces blank.

This section discussed the design of the data structures and the information they contain. This information is used to generate output products, and the information is stored in files. These files of information are discussed in the next section.

Data Files

Like the data structures used by the tool, the data files each have their own format that is different from Urscheler's prototype. The tool is capable of creating five different data files, two that save the raw data (graphics and data dictionary information) and three that save the output products from the tool. The following paragraphs briefly describe the files and their formats.

The graphics information from the diagram is maintained in an ASCII file labeled as <filename>.gph. In the file, squiggle line, footnote, box, and line information is saved in a precisely formatted manner. If a field has been left blank by the user, a "\$\$NULL\$\$" string is placed in the file as a place holder. Appendix C contains an exact definition of how this file is stored.

The data dictionary information from the diagram is maintained in an ASCII file labeled as <filename>.dbs. The data dictionaries for each activity and only the data dictionary data elements defined by the user are saved in this file. The design of this file format is the result of another thesis in progress at the time of this writing

(Connally, 1987:). This format is also defined in Appendix C.

The facing page text for the diagram is saved (at the option of the user) in an ASCII file labeled as <filename>.fpt. The facing page text is a copy of the description fields of the data dictionaries for each activity box. The format in which it is saved is specified in Appendix A.

The user also has the option of saving in a file a formatted copy of all or selected data dictionaries. This file is an ASCII file that is labeled as <filename>.dd. The format for this type of file is also specified in Appendix A.

The user has the option of saving a copy of the diagram generated. This file contains a SUN raster image of the diagram that is labeled as <filename>.dmp. SunView raster functions are used to read the diagram image from the screen and save it in the SUN's raster file format. The user may obtain hardcopy by using the UNIX "lpr -v" command.

Coding Approach

Having completed the design specification, defined the data structures, and defined the file formats, the next step was the actual coding of the modules. As previously mentioned, the programming language used to implement this tool was C. This section describes the coding approach.

A top down approach (Pressman, 1982:232) was taken in developing the modules. The process started by developing the main control module that called the appropriate functions associated with each menu selection. With the control structure and numerous module "stubs" in place, the module stubs were replaced with the real code and each stub's lower level modules until the project was complete.

Internal documentation of the code followed that prescribed in the AFIT/ENG Software Development Documentation Guidelines and Standards. Each file began with the standard file header (Hartrum, 1986:38) and each module began with the standard module header (Hartrum, 1986:40). In addition, C language comments were provided in the code to amplify and clarify sections of the code.

Testing Approach

Testing was accomplished in conjunction with the coding phase in this project. A slightly "impure" integration test method was applied (Pressman, 1982:298-9). The procedure was impure because neither the depth-first or breadth-first incorporation approach to integrating new software into the existing software was consistently applied due to the time constraints of the project. Typically, about ten modules were written and added to the software before applying the tests.

Obviously, an exhaustive test was impossible for a project of this magnitude; however, the tests conducted

attempted to assure that all module paths were exercised at least once, that all conditional statements were verified, that each module provided the needed function, and that errors were properly handled. To assist future maintenance actions on the code, debugging "printf" statements identifying its position in the code were used to alert the user when an unanticipated condition did occur.

Summary

This chapter discussed the design decisions based on the requirements and objectives of the thesis. Five reasons why the SUN workstations met the hardware requirements were stated. Second, portability of the tool, reusability of the Urscheler code, and suitability of the SunView environment were issues covered by the software considerations. Considerations from previous studies found data dictionary constraints that were maintained in this tool and found design decisions that were carried over from past works. Next, the screen layout, menu system, and use of voice were identified as the key human/computer interface topics that concerned this tool. Also, decisions involved in the design of the data structures and data files were identified. Finally, coding and testing approaches were explained in the last two parts of the section. Given this information, the next chapter discusses the tool's operation and the results of the tool's evaluation.

V. SA TOOL OPERATION AND EVALUATION

The issues and decisions surrounding the design of the tool were discussed in the previous chapter. The purpose of this chapter is to provide a broad overview of how the tool operates and to report the results of the tool's evaluation by a group of AFIT students.

This chapter will first discuss the tool and its function and then report the evaluation results. The first section presents the tool's function in six parts: initialization, graphics functions, data dictionary functions, facing page text functions, input/output functions, and miscellaneous functions. The user is referred to the User's Reference Manual (Appendix J) for a complete discussion of all the available functions. The first part discusses the initialization of the tool.

Operation

Initialization. To start the tool running, the user must first invoke the Suntool environment. This is accomplished by entering the command "sunttools" at the UNIX prompt. Once a C-shell window has been opened, the user is ready to run the tool.

The command to start the tool is "SAtool." The tool was designed to accept a -v option to enable the use of a

DECTALK voice synthesizer to provide audio feedback for the user; however, this feature was not implemented due to the time constraints involved in the project. The user is now ready to begin using the tool and may select one of the oval buttons in the Selection Window for a menu of choices.

Graphics Functions. The graphics functions, located on menus under the "EDIT DIAGRAM" oval, contain all the functions needed to draw and label an SA diagram. The functions make it possible to add a new or edit an existing graphical entity (activity box, ICOM line, squiggle line, diagram label, or footnote marker).

Five operations are permitted on activity boxes. Adding an activity box involves setting the location of the box as well as entering the box name and box number. To edit one of the box attributes, the user identifies which box by placing the cursor inside the box and clicking a mouse button. Each of the operations involved in adding a box, setting the location and entering the name and number, may be changed by picking the appropriate menu selection. Finally, an existing activity box may be deleted.

Six operations are permitted on lines. Adding a line involves selecting the two endpoints using the mouse and selecting the begin and end attributes from their respective menus. If the end attribute is a Branch or Turn, the tool automatically positions the cursor at the end of the Branch or Turn to continue drawing the line. To add or

change a line label, the user selects the menu function, "Edit Line Label," and moves the cursor to a point on the line and clicks the mouse to select the line. The text is then entered from the keyboard. A TO-ALL, FROM-ALL, or ICOM label, specified when a line is added, may be changed by selecting "Edit TO/FROM Label" or "Edit ICOM Label." Finally, the line may be deleted entirely or redrawn with the "Redraw/Delete Line" selection. The choice of these two operations is made by clicking a mouse button. The Redraw function deletes a line and any connected to it and starts the user re-drawing the line with the original beginning attributes.

Fifteen operations are permitted on the seven diagram header labels. Each diagram header label may be first added, or later edited, for a total of fourteen. The fifteenth operation, useful at the start of a new diagram, allows the addition of all seven header fields without selecting each operation individually.

Four operations are permitted on footnotes. Since the footnotes are created and maintained in pairs of identical boxes, creating a footnote box pair involves first specifying the number label for both boxes, then moving each box to its desired location. Changing the footnote number will change the number in both boxes while changing a footnote location moves only one of the pair at a time. To delete a footnote box pair, the user places the

cursor inside one of the footnote boxes and clicks a mouse button; both boxes are then deleted from the diagram.

Two operations are permitted on squiggle lines, creating and deleting. To create a squiggle line, the user is allowed to draw three, "free-hand" line segments. To delete a squiggle line, the appropriate squiggle line is selected by placing the cursor on a point on one of the line segments and clicks a mouse button. The squiggle line is then removed from the diagram.

Having considered the functions needed to create and edit the SA diagram entities, the next paragraphs discuss the functions needed to complete the data dictionary information.

Data Dictionary Functions. The data dictionary functions, located on menus under the "EDIT DD" oval, contain all the functions needed to enter and edit any data dictionary information for the diagram. The information is entered via the Data Dictionary Window.

There are five operations available for managing the data dictionary information. "ADD DD" is used to create storage for data dictionary information for a line. The user selects the line by moving the cursor on the line and clicking a mouse button. The data dictionary template is placed in the Data Dictionary Window with the number in parentheses before the field name specifying the width of the field. While entering the information, the "ADD MORE

ALIASES" selection adds another alias block to the existing template. Reviewing or further editing an existing data dictionary is the purpose of the "EDIT DD" function. (Recall that data dictionaries for activity boxes are automatically generated.) Selecting "DD-FINISHED" causes the information in the Data Dictionary Window to be collected and stored in the data dictionary data structures. Finally, "SAVE DD IN FILE" permits the user to select one or more data dictionaries or all data dictionaries for saving in an ASCII file in the format specified in Appendix A.

Facing Page Text Functions. Located under the "FPT FUNCTIONS" oval are two facing page text functions. Both operations build the facing page text according to the format in Appendix A from the descriptions entered in the data dictionary. The first function displays this information in the Data Dictionary Window while the second function puts the information in an ASCII file.

Input/Output Functions. The input and output functions are located under the "RECALL DIAGRAM" and "SAVE DIAGRAM" ovals, respectively. A previously saved diagram may be read in with the "RECALL DIAGRAM" function by specifying the filename (without a dot (.) extension). The program checks the working directory for files named with the given file name plus ".dbs" and ".gph" extensions and loads their information into the data structures.

The diagram currently stored in the tool's data structures may be saved in a similar manner, with the tool adding the dot extensions to the file name. The user selects either "Save for DB" to create a ".dbs" file for saving in a relational database or "Save Local" to create a file with a ".dbs" file that is not compatible with the database tool, yet remains compatible with this tool.

Miscellaneous Functions. The miscellaneous functions, found under the "MISC FUNCTIONS" oval, contain operations to save a file that generates a hardcopy of the diagram, to manipulate the working directory, to erase the current diagram and data dictionary, to redisplay the Diagram Window, and to quit the tool. "Make Diagram (Normal)" and "Make Diagram (Sideways)" may be selected depending on the desired orientation of the SA diagram on the page. The user specifies a file name and the tool adds a ".dd" extension to the file that can be printed on SUN laser printer. The current working directory is displayed in the Message Window by selecting "Display Directory" or can be changed by selecting the "Change Directory" function. "Start New Diagram" erases the Diagram Window and empties the data structures of the information being stored. The "Redisplay Diagram" and "QUIT" functions are self explanatory.

Given this broad overview of the tool and its operation, the next section discusses the results of an evaluation of the tool.

Evaluation Results

Two graduate software engineering classes at AFIT used the tool in conjunction with a homework problem and evaluated their experiences with the tool. In this section, the evaluation methodology is described followed by the conditions surrounding the evaluation. Finally, the results of the evaluation are presented.

Evaluation Methodology. The student's reactions to the tool were gathered using a quantitative questionnaire developed at AFIT (Mallary, 1985: 81-5; Foley:1986). For the purposes of this evaluation, the use of this method was assumed valid. The questionnaire consisted of 12 questions regarding different aspects of the tool. Figure 9 is an example of one of these questions. Figure 10 is a list of the 12 questions. Question 12 gathers an overall rating of satisfaction that was compared to the average of the other 11 questions.

Each of the first 11 questions was rated on a scale from -3 to 3 on four adjective-pairs. For each user i and question j , these four scores were averaged to give a reaction score R_{ij} . Each question also had an overall satisfaction score for correlation with the average of the adjective-pair score (unused in this evaluation). Finally, to weight the adjective-pair score (W_{ij}), each question allowed the user to rate the importance of the particular factor to him on a scale of 0 to 1.

From this information, a normalized satisfaction score NS for each user was computed as

$$NS = 1/(3 \cdot N_i) \sum_{j=1}^{11} R_{ij} \cdot W_{ij}$$

where N_i is the number of questions whose reaction score R_{ij} was not zero. The normalized satisfaction score indicates the degree of user satisfaction according to Table 1.

1. System Feedback: The extent to which the system kept you informed about what was going on in the program.

insufficient | | | | | | | | | | | sufficient

unclear | | | | | | | | | | | clear

useless | | | | | | | | | | | useful

bad | | | | | | | | | | | good

unsatisfactory | | | | | | | | | | | satisfactory

To me this factor is

unimportant | | | | | | | | | | | important

Comments:

Figure 9. Example Evaluation Question Format
Source: (Mallary, 1985:111)

Evaluation Conditions. Thirty-three students were assigned to make one diagram using the tool and complete the evaluation based on that experience. Each student was given a guide that explained the workstation's login and logout

TABLE 1

Bailey and Pearson Ratings

NORMALIZED SCORE	TRANSLATION
+1.00	Maximally satisfied
+0.67	Quite satisfied
+0.33	Slightly satisfied
+0.0	Neutral
-0.33	Slightly dissatisfied
-0.67	Quite dissatisfied
-1.00	Maximally dissatisfied

Source: (Mallory, 1985)

procedures, error recovery procedures, menu selection procedures, and the components of the screen layout. Information regarding the exact function provided by each menu selection was not available to give to the first class of students; therefore, it was not provided to the second class to maintain consistency across the two classes.

Evaluation Results. The average of the normalized scores for the first 11 questions of the evaluations was 0.295 or about "Slightly Satisfied" according to the translation of Table 1. The scores ranged from -0.182 to .682 with a standard deviation from the mean of 0.294. This compares closely with an average 0.333 "overall satisfaction" rating given on question 12. On the average, the evaluators used the tool about 138 minutes. Table 11 shows how the evaluation broke down by question.

1. System Feedback or Content of the Information Displayed. The extent to which the system kept you informed about what was going on in the program.
2. Communication. The methods used to communicate with the tool.
3. Error Prevention. Your perception of how well the system prevented user induced errors.
4. Error Recovery. The extent and ease with which the system allowed you to recover from user induced errors.
5. Documentation. Your overall perception as to the usefulness of the documentation.
6. Expectations. Your perception as to the services provided by the system based on your expectations.
7. Confidence in the System. Your feelings of assurance or certainty about the services provided by the system.
8. Ease of Learning. Ease with which you were able to learn how to use the system to generate IDEF0 definitions.
9. Display of Information. The manner in which both program control and IDEF0 information was displayed on the screen.
10. Feeling of Control. Your ability to direct or control the activities performed by SATool.
11. Relevancy or System Usefulness. Your perception of how useful the system is as an aid to a software developer.
12. Overall Evaluation of the System. Your overall satisfaction with the system.

Figure 10. Evaluation Questions

TABLE 11

Average Normalized Score by Question

QUESTION	AVERAGE NORMALIZED SCORE
1	0.453
2	0.399
3	0.060
4	-0.056
5	-0.055
6	0.318
7	0.406
8	0.406
9	0.453
10	0.346
11	0.475
12	0.333

The lack of adequate documentation of the tool for the evaluation probably contributed heavily to the lower scores received on questions 3, 4, and 5. One user commented, "I would rather read and see example displays, as opposed to the trial and error method (of learning the tool)." With regard to question 5, the users frequently commented they could find no documentation, and four users failed to even rate the question. Regarding question 4, users described mistakes they made in drawing the diagram, noting they were unable to undo the error. In every case there was a menu selection available to undo the error, but the person was apparently unaware of the selection's existence. By the same reasoning regarding question 3, the users probably made more errors experimenting with the menu selections trying to

determine their functionality, leaving an impression of poorer than expected error prevention capability.

The users offered suggestions for improving the tool. For instance, one suggested that a help selection be added to each of the menus. This would be a useful and straightforward addition to the tool. Also, it was suggested that a universal undo command for each menu selection be made available. This would also be useful, but much less straightforward to implement.

Summary

This chapter discussed the operation of the tool and an evaluation of the tool. It gave procedures for invoking the Suntool environment and the SAtool. Each operation permitted on the five major graphical entities (activity boxes, ICOM lines, squiggle lines, diagram labels, and footnote markers) was identified and described. Next, the data dictionary capabilities of the tool were described followed by the facing page text functions. Following this description was a discussion of the input/output functions. Finally, some miscellaneous functions provided by the tool were described.

An evaluation of the tool was performed using a quantitative questionnaire and statistical analysis of the results. The method of computing the normalized satisfaction score was detailed as well as the conditions under which the experiment was conducted. The results found

that users were "slightly satisfied" with the performance of the tool.

The following chapter presents the researcher's conclusions from conducting this thesis and recommends several future projects as a result of this effort.

VI. CONCLUSIONS AND RECOMMENDATIONS

Conclusions

The purpose of this thesis was to specify and design a tool that allows a user to interactively create and edit structured analysis diagrams. In addition, partial data dictionary information is automatically generated from graphics information by the tool and is supplemented by inputs from the user.

Each of the three phases of this effort was accomplished successfully. During the first phase, the Urscheler implementation was analyzed and the reusable parts of his software were identified. Also, a graphical SA language was derived from several sources and a subset identified for implementation in this tool. During the second phase, the structured analysis and data dictionary methodologies were combined and the tool was built and successfully tested. During the third phase, the usefulness of the tool was evaluated by polling people who used the tool for a classroom software engineering project.

The evaluation found the users were "slightly satisfied" with the performance of the tool. Three questions rating the documentation, error prevention, and error recovery capabilities were given significantly lower ratings because no documentation of each function provided the tool was available at the time of the experiment.

Although the objectives of this thesis were accomplished, there are several aspects of the tool that could be enhanced to make it a better product. These aspects are enumerated in the next section.

Recommendations

The recommendations are divided into two categories: small scale projects and large scale projects. Presented first are the small scale projects.

Small Scale Projects. The following are recommended small scale projects for improving the tool:

1. Integrate the use of voice. This project should go back and analyze the "put message()" functions that provide the user with feedback in the tool's Message Window. This function was designed to send messages either to the Message Window or a Dectalk voice synthesizer or both. Currently, all messages are routed to the Message Window. Developing this capability could enhance the human/computer interface.
2. Improve the method for making hardcopies of the diagram. The current method reproduces a pixel by pixel image of the Diagram Window into a SUN rasterfile. A method should be considered that would command the printer to draw lines and text rather than generating pixel by pixel images.
3. Improve the method by which the user inputs data dictionary information. The current method uses a generic editor that is unable to prevent the user from entering information past the end of the different length fields. If the information exceeds the field boundary, the information is truncated when it is saved. A method should be investigated that will not allow the user to enter information beyond the end of the field.

4. Compatibility test this tool with Foley's tool via the relational database. The data dictionary information for this tool was designed to match that of the Foley editor. Testing should be conducted to ascertain the compatibility of the two tools.
5. Provide on-line help. Currently, help from the tool is provided via the Message Window. Due to the size of the window, help messages tend to be cryptic. Therefore, a method of providing more in-depth explanations of commands from the tool should be considered. Adding one "help" selection to each menu has been suggested.

Large Scale Projects. The following are recommended large scale projects for improving the tool:

1. Convert the tool from SunView to a standard graphical package like GKS. Converting to a graphics standard would increase the possibility of porting the tool other workstations.
2. Make the tool capable of handling an entire project. This would require adding more data structures to maintain the graphical and data dictionary information for all the diagrams in a given project. This enhancement would allow more consistency checking algorithms for the data and allow the tool to generate more complete data dictionaries for data elements. Also, a method to walk through the diagram hierarchy would need to be implemented. Examining current commercial tools and their "explode" capability, the ability to see an activity box's decomposition by exploding the box for a "closer" look inside, might be helpful.
3. Remove the necessity of maintaining graphical information. This would require an analysis of the data dictionary format to ensure it is capable of holding all the information contained in a graphical picture. Once this is proven, it should be possible to automatically draw the diagram from the information contained in the data dictionary.

4. Add color to enhance the human/computer interface. Colors might be used to reflect the presence of data dictionary information for a line or to reflect the presence of poor coupling and cohesion characteristics among the activity boxes to suggest just a few of the possibilities. Here, a trade-off with portability should be carefully considered.
5. Redo the entire project in Ada. This project would be contingent upon the upgrade of SunView to support Ada. This project would make the tool better suited for use by other agencies in the U.S. government, given its requirement for using Ada for all new software developments.

Appendix A: AFIT STRUCTURED ANALYSIS SYNTAX

The purpose of this appendix is to precisely define the formats of the documents generated by the structured analysis tool. The AFIT syntax for structured analysis diagrams is derived from SADT and IDEF₀ graphic syntax. Also, the format for the data dictionary entries, and the format for facing page text is specified.

SADT and IDEF₀

A tabulation of the SADT language's syntax and concepts was presented in a paper by Douglas Ross in 1977 (Ross, 1977). Figure 11 is the figure presented in that paper.

The subset of the SADT syntax used by IDEF₀ was defined in the "User's Reference Manual," published by the Materials Laboratory at Wright-Patterson AFB, Ohio (IDEF₀ Manual, 1981). Some of the language used in example diagrams was not described in the text. Therefore, Figure 12 was generated to summarize the IDEF₀ syntax in a table based on the example diagrams and text in IDEF₀ user's manual. Additionally, the column called "term" was the name by which each notation was referred. Some entries in the Ross article table are methods and not graphical entities; therefore, they cannot be implemented in a CAD tool. An example of this is Line 36 in Figure 11.

Not all the graphical entities presented by Ross were implemented in the IDEF₀ language. Figure 13 is a summary of those notations.

	PURPOSE	CONCEPT	MECHANISM	NOTATION	MODE
1	BOUND CONTEXT	INSIDE/OUTSIDE	SA BOX		A21
2	RELATE/CONNECT	FROM/TO	SA ARROW		A22
3	SHOW TRANSFORMATION	INPUT/OUTPUT	SA INTERFACE		A23
4	SHOW CONSTRAINTS	CONTROL	SA INTERFACE		A24
5	SHOW REASON	SUPPORT	SA MECHANISM		A25
6	NAME APPLY	ACTIVITY DATE HAPPENING THING	SA NAMES		A26
7	LABEL APPLY	THINGS HAPPENINGS	SA LABELS		A27
8	SHOW NECESSITY	IFC	PATH		A28
9	SHOW DOMINANCE		CONSTRAINT		A29
10	SHOW RELEVANCE	ICC	ALL INTERFACES		A30
11	DRIFT OBVIOUS	END	DRIFTED ARROW		A31
12	BE EXPLICIT WITHOUT CLUTTER	PIPELINES, CONDUITS, WIRES	BRANCH		A32
13			JOIN		A33
14	BE CONCISE AND CLEAR	CABLES, MULTI-WIRES	BUNDLE		A34
15			SPREAD		A35
16	SHOW EXCLUSIVES	EXPLICIT ALTERNATIVES	OR BRANCH		A36
17			OR JOIN		A37
18	SHOW INTERFACES TO PARENT DIAGRAM	ARROWS PENETRATE	SA BOUNDARY ARROWS (ON CHILD)		A38
19	SHOW EXPLICIT PARENT CONNECTION	NUMBER CONVENTION FOR PARENT, WRITE ICOM CODE ON CHILD, BOUNDARY ARROWS	NUMBER AND PAGE NUMBER OF DATA DIAGRAM		A39
20	SHOW UNIQUE DECOMPOSITION	DETAIL REFERENCE EXPRESSION (DRE)	SA CALL ON SUPPORT		A40
21	SHOW SHARED OR VARIABLE DECOMPOSITION				A41

	PURPOSE	CONCEPT	MECHANISM	NOTATION	MODE
22	SHOW COOPERATION	INTERMEDIATE OF SHARED RESPONSIBILITY	SA 2-WAY ARROW		A42
23	SUPPRESS INTERMEDIATE DETAIL	ALLOW ONLY WAY WITHIN WAY REPRESENT	TAKE OUT THE BUTTER ARROWS		A43
24	SUPPRESS TRAIL-THROUGH DETAIL	ALLOW ARROWS TO GO OUTSIDE DIAGRAM	SA TRAIL-THROUGH WITH REFERENCES		A44
25	SUPPRESS NEEDED ARROW CUTS	ALLOW TRAIL CUTS WITHIN DIAGRAM	TO ALL BY FROM ALL		A45
26	SHOW REEGL ANNOTATION	ALLOW WORDS IN DIAGRAM	SA NOTE		A46
27	OVERCOME CRAMPED SPACE	ALLOW REMOTE LOCATION OF WORDS IN DIAGRAM	SA FOOTNOTE		A47
28	SHOW COMMENTS ABOUT DIAGRAM	ALLOW WORDS OUTSIDE IN DIAGRAM	SA META-NOTE		A48
29	ENSURE PROPER ASSOCIATION OF WORDS	USE WORD TO INTENDED SUBJECT	SA "SINGLE-POINT"		A49
30	UNIQUE SHEET REFERENCE	CHRONOLOGICAL CREATION	SA NUMBER		A50
31	UNIQUE BOX REFERENCE	PATH DOWN TREE FROM BOX NUMBERS	SA MODE NUMBER (BOX NUMBERS)		A51
32	SAME FOR MULTI-MODELS	PRECEDE BY MODEL NAME	SA MODEL NAME		A52
33	UNIQUE INTERFACE REFERENCE	ICOM WITH BOX NUMBER	SA BOX ICOM		A53
34	UNIQUE ARROW REFERENCE	FROM TO	PAIR OF BOX ICOMS		A54
35	SHOW CONTEXT REFERENCE	SPECIFY A REFERENCE POINT	SA REF EXP "DO"		A55
36	ASSIST "CORRECT" INTERPRETATION	SHOW DOMINANCE GEOMETRICALLY ASSIST PARSING	STAIRCASE LAYOUT		A56
37	ASSIST UNDERSTANDING	PROSE SUMMARY OF MESSAGE	SA TEXT		A57
38	HIGHLIGHT FEATURES	SPECIAL EFFECTS FOR EXPOSITION ONLY	SA FEOS		A58
39	DEFINE TERMS	GLOSSARY WITH WORDS & PICTURES	SA GLOSSARY		A59
40	ORGANIZE PAGES	PROVIDE TABLE OF CONTENTS	SA MODE INDEX		A60

Figure 11. SADT Language Features
Source: (Ross, 1977:20)

Ross article line number	Term	User's Manual Reference
1	BOX	2-2,3
2	ARROW	2-2,3
3	INPUT	3-26 (FIG)
3	OUTPUT	3-26 (FIG)
4	CONTROL	3-26 (FIG)
5	MECHANISM	3-11
6	ACTIVITY NAME	2-3,4
7	LABEL	2-3,4
12	BRANCH	3-9
13	JOIN	3-9
14	BUNDLE	6-14
15	SPREAD	6-14
18	BOUNDARY ARROW	3-17
20	DETAILED REFERENCE NUM	2-3
22	2-WAY ARROW	3-26 (FIG)
24	TUNNEL ARROW	2-3
25	TO/FROM ALL	6-21
27	FOOTNOTE	3-26 (FIG)
28	META-NOTE	2-3
29	SQUIGGLE	3-26 (FIG)
30	C-NUMBER	2-3
31	NODE NUMBER	2-3
32	MODEL NAME	3-26 (FIG)
33	ICOM CODE	4-8
37	FACING PAGE TEXT	4-1
38	FEO (FOR EXPOSITION ONLY)	6-5
39	GLOSSARY	2-3
40	NODE INDEX	2-3

Figure 12. IDEF₀ Graphic Syntax

Ross article line number	Term	User's Manual Reference
16	OR branch	NOT FOUND
17	OR join	NOT FOUND
21	CALL ON SUPPORT	NOT FOUND
23	2-1 WAY BUTTING ARROW	NOT FOUND
26	NOTE	NOT FOUND
36	REF. EXP. "DOT"	NOT FOUND

Figure 13. Graphic Notations Unused by IDEF₀

An additional graphical feature is used in both the Softech and IDEF₀ diagrams but not described by IDEF or the Ross article, is the use of the slash (/) to separate the forward and backward content of double headed arrows. This syntax is described in other Softech literature describing the SADT methodology (Softech Inc., 1976:4-21).

AFIT Structured Analysis Diagram Syntax

A sampling of structured analysis diagrams generated by AFIT students and faculty found the syntax adopted by IDEF₀ is sufficient for performing requirements analysis. None of the graphic notations described by Ross and unused by IDEF₀ were found in the diagrams sampled. For these reasons, Figure 12 is also the AFIT syntax.

Subset of AFIT Syntax Implemented

Figure 14 is the subset of the AFIT structured analysis syntax that was implemented by the tool. Due to the time limitation for implementing the tool, some of the low priority constructs were not implemented. Presented next are the reasons the five graphical constructs were not implemented by this tool.

Meta-notes are additional comments about a structured analysis diagram and are placed on the diagram. This feature was not implemented because the diagrams must be tied to the data dictionaries as much as possible.

Ross article line number	Term	User's Manual Reference
1	BOX	2-2,3
2	ARROW	2-2,3
3	INPUT	3-26 (FIG)
3	OUTPUT	3-26 (FIG)
4	CONTROL	3-26 (FIG)
5	MECHANISM	3-11
6	ACTIVITY NAME	2-3,4
7	LABEL	2-3,4
12	BRANCH	3-9
13	JOIN	3-9
18	BOUNDARY ARROW	3-17
22	2-WAY ARROW	3-26 (FIG)
24	TUNNEL ARROW	2-3
25	TO/FROM ALL	6-21
27	FOOTNOTE	3-26 (FIG)
29	SQUIGGLE	3-26 (FIG)
30	C-NUMBER	2-3
31	NODE NUMBER	2-3
32	MODEL NAME	3-16
33	ICOM CODE	4-8
37	FACING PAGE TEXT	4-1

Figure 14. Implemented Graphic Syntax

Meta-notes do not correlate to any data dictionary entry.

According to Ross:

There is no way that information in metanotes can participate in the information content of the diagrams, and therefore they should not be used in an attempt to affect the interpretation of the diagrams themselves, but only for mechanical operations regarding the diagram's physical format or expression (Ross, 1977:30).

Bundle and spread were not implemented because these constructs can be represented as a special case of the

Join and Branch, respectively. Implementing both sets of constructs in the tool would be a duplication of effort. FEO's pictorially highlight features and special effects of the diagram. They were not implemented by this tool because they do not correspond to any data dictionary entry, except the description. The user should attempt to highlight features of the diagram in words in the facing page text of the diagram.

The purpose of the Glossary is to define terms using words and pictures. Again, because items normally found in the Glossary cannot be directly tied to the data dictionary, it was not implemented.

Data Dictionary Formats

There are two types of data dictionary formats, one to describe activities and one to describe data elements. For this tool, the data dictionaries generated were of the formats specified by the AFIT Software Development Documentation Guidelines and Standards (Hartrum, 1986).

Figure 15 shows the format for the information inserted into a Data Dictionary for Activity. Figure 16 shows a completed example of this type of data dictionary entry.

Figure 17 shows the format for the information inserted into a Data Dictionary for Data Element. Figure 18 shows a completed example of this type of data dictionary entry.

NAME: (of activity)
TYPE: ACTIVITY
PROJECT: (Project name)
NUMBER: (Node number of this activity)
DESCRIPTION: (Multiple lines allowed)
INPUTS: (Multiple lines allowed-one entry/line)
OUTPUTS: (Multiple lines allowed-one entry/line)
CONTROLS: (Multiple lines allowed-one entry/line)
MECHANISMS: (Multiple lines allowed-one entry/line)
ALIASES: (Names of aliases, multiple lines allowed)
 COMMENT: (Why is this alias needed?)
PARENT ACTIVITY: (Name of parent activity)
RELATED REQUIREMENT NUMBER: (Paragraph number of
 textual requirements statement)
 (Multiple lines allowed-one entry/line)

VERSION: (version of this data dictionary entry)
VERSION CHANGES: (Why was the last version updated?)
DATE: (of this entry)
AUTHOR: (of this entry)

Figure 15. Format of a Data Dictionary for Activity

```

1
NAME:  MANIPULATE FILES
TYPE:  ACTIVITY
PROJECT:  NETOS
NUMBER:  A12
DESCRIPTION:  This activity handles all remote
requests to manipulate actual files.  This
includes printing local files, storing local files
on the MSS, and requesting files from the MSS.
INPUTS:
    KEYBOARD INPUT
    MSGS & DATA
    DISK FILES
OUTPUTS:
    CRT OUTPUT
    MSGS & DATA
    DISK FILES
    DONE
CONTROLS:
    SELECTION
MECHANISMS:
ALIASES:
    COMMENT:
PARENT ACTIVITY:  EXECUTE REMOTE FUNCTION
RELATED REQUIREMENT NUMBER:  1.2.1

VERSION:  1.0
VERSION CHANGES:
DATE:  9/13/81
AUTHOR:  T. C. Hartum

```

figure 16. Data Dictionary for Activity

```

NAME: (of this data item)
TYPE: DATA ELEMENT
PROJECT: (Project name)
DESCRIPTION: (Multiple lines allowed)
DATA TYPE: (if known)
MIN VALUE:(if applicable-multiple lines allowed-1/line)
MAX VALUE:(if applicable-multiple lines allowed-1/line)
RANGE: (if applicable)
VALUES: (allowable values, if appropriate - multiple
        lines allowed-1 entry per line)
PART OF: (parent data element - multiple lines
        allowed-1 entry per line)
COMPOSITION: (sub elements, if any. multiple lines
        allowed - 1 entry per line)
ALIASES: (Names of aliases, multiple lines allowed)
    WHERE USED: (IDEF0 activity number)
    COMMENT: (Why is this alias needed?)
SOURCES: (IDEF0 activity names)
DESTINATIONS: (IDEF0 activity names)
RELATED REQUIREMENT NUMBER: (Paragraph number of
        textual requirements statement)
        (Multiple lines allowed-one entry/line)

VERSION: (version of this data dictionary entry)
VERSION CHANGES: (Why was the last version updated?)
DATE: (of this entry)
AUTHOR: (of this entry)

```

Figure 17. Format of a Data Dictionary for Data

1
NAME: MSGS & DATA
TYPE: DATA ELEMENT
PROJECT: NETOS
DESCRIPTION: NETOS messages and data, including files,
transferred around the network.
DATA TYPE:
MIN VALUE:
MAX VALUE:
RANGE:
VALUES:
PART OF:
COMPOSITION:
 MSGS
 DATA
ALIASES: messages and data
WHERE USED: A122, A135
COMMENT: Due to group's inconsistency
SOURCES:
 MANIPULATE FILES
 MANIPULATE FILE INFORMATION
DESTINATIONS:
 MANIPULATE FILES
 MANIPULATE FILE INFORMATION
 MANIPULATE COMMO LINKS
RELATED REQUIREMENT NUMBER: 1.2.4.2

VERSION: 1.0
VERSION CHANGES:
DATE: 9/13/84
AUTHOR: T.C. Hartrum

Figure 18. Data Dictionary for Data

Facing Page Text Format

The format for the facing page text generated by this tool was also based on a format provided in AFIT's Software Development Documentation Guidelines and Standards. The AFIT data dictionaries are designed to contain all the information provided in the structured analysis diagram and its facing page text. To map the facing page text to the data dictionary, the facing page text must match exactly the text in the "Description" field of that activity's data dictionary. The format and rules for facing page text are shown in Figure 19.

1. A heading is located at the left column and consists of the Node number, then two spaces, and then the Title (exactly as given on the diagram).
2. Following the headings is a section beginning with "Abstract:," then two spaces, followed by an overview of the diagram. If a parent diagram for this diagram exists, then the text matches exactly the description given on the parent diagram facing page text.
3. For each activity box in the diagram and in order according to the node number, a paragraph starting with the node number, followed by two spaces, and a description of the activity follows the abstract. This description will match exactly the "Description" field of the data dictionary.
4. The lines between the heading, the abstract, and each activity paragraph are double spaced.
5. The first line of each activity paragraph is indented 5 spaces.

Figure 19. Facing Page Text Format

Figure 20 is an example of facing page text generated according to this format.

A0 Provide Requirement Analysis CAD Tool

Abstract: Provide Requirement Analysis CAD Tool provides the user a mechanism by which he is able to draw activity SA diagrams. From these diagrams, facing page text and Data Dictionaries for Activities and Data are generated.

A1 Create SA Diagrams gathers the user inputs and converts them into graphical information representing the diagram. Throughout the process, the user is informed of progress through updates on the CRT screen. The user may also use an existing file of user inputs to generate a diagram. Data structures are created from which inputs to the data dictionaries may be gathered.

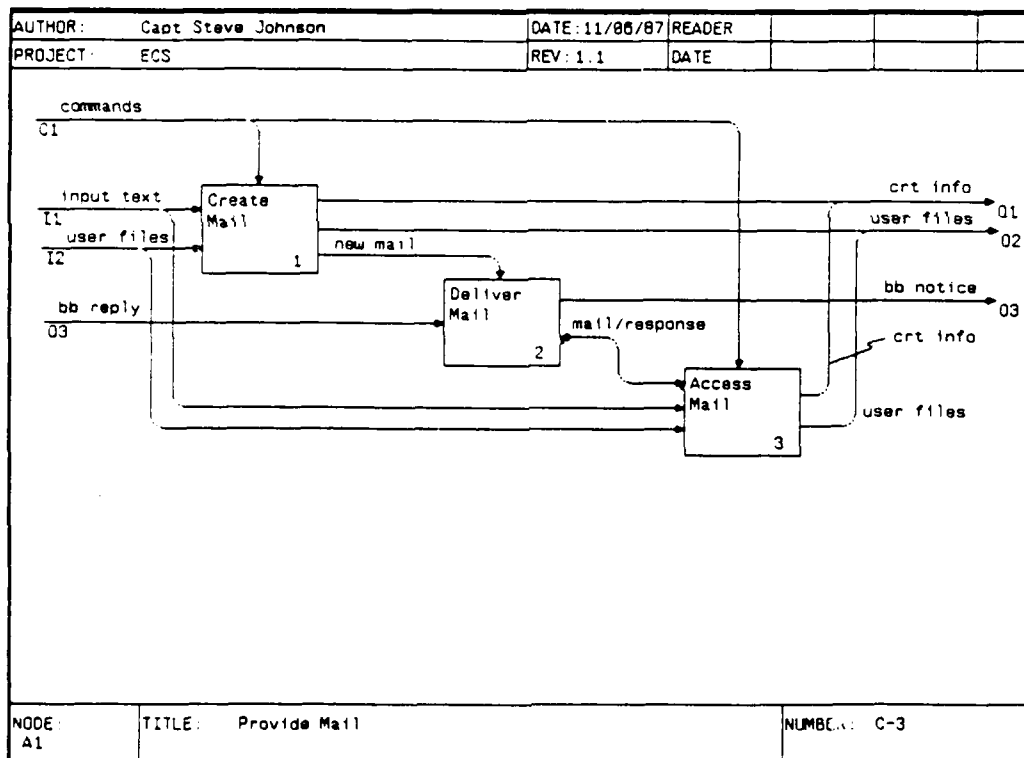
A2 Create DD takes the graphical information pertinent to the data dictionary (activity or data) along with user inputs to create the appropriate data dictionary according to the AFIT data dictionary format. The status of the process is maintained on the CRT.

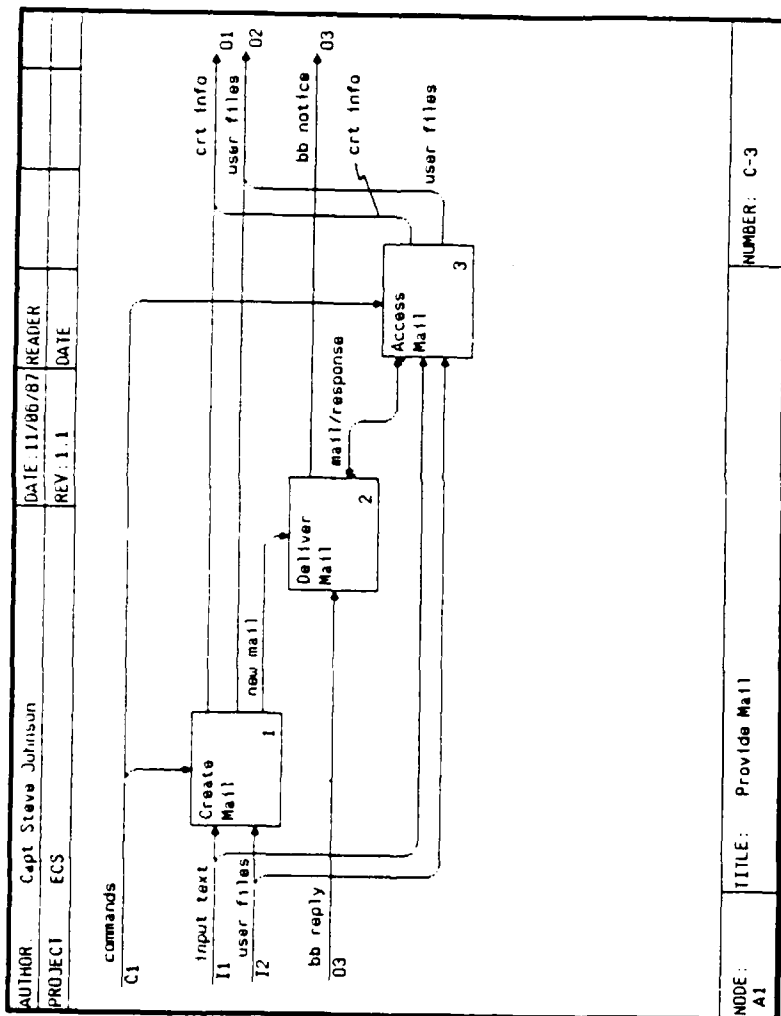
A3 Create Facing Page Text combines the description field of the data dictionaries for activities with the changes desired by the user to produce a facing page text. The status of the process is maintained on the CRT.

Figure 20. Correctly Formatted Facing Page Text

Appendix B: Example Outputs

The following pages contain example outputs of the tool. The first page is an example of an SA diagram generated using "Make Diagram (Normal)" selection. The second page is an example of an SA diagram generated using the "Make Diagram (Sideways)" selection. The third page is an example of an Activity Element Data Dictionary and the fourth page is an example of a Data Element Data Dictionary. Finally, the fifth page is an example of the Facing Page Text for a diagram.





|
NAME :Access Mail
TYPE :ACTIVITY
PROJECT :ECS
NUMBER :A13
DESCRIPTION :This activity takes inputs and user files
and parses user's commands to allow access to his mail.
INPUTS :
 input text
 user files
 mail
OUTPUTS :
 response
 crt info
 user files
CONTROLS :
 commands
MECHANISMS :
ALIASES :
COMMENT :
PARENT ACTIVITY :Provide Mail
RELATED REQUIREMENT NUMBER :
VERSION :1.1
VERSION CHANGES :Changed "bulliten board" to bb.
DATE :11/06/87
AUTHOR :Capt Steve Johnson

|
NAME : new mail
TYPE : DATA ELEMENT
PROJECT :ECS
DESCRIPTION :New Mail is a new message generated by
the user that includes the appropriate header information
as well as the body of the message.
DATA TYPE :
MIN VALUE :
MAX VALUE :
RANGE :
VALUES :
PART OF :
COMPOSITION :
ALIASES :none
WHERE USED :
COMMENT :
SOURCES :

DESTINATIONS :

RELATED REQUIREMENT NUMBER :
VERSION :1.1
VERSION CHANGES :changed "bulliten board" to bb.
DATE :11/06/87
AUTHOR :Capt Steve Johnson

A1 Provide Mail

Abstract: Provide Mail allows the user to build a message, access any message, and transmit any message.

A11 This activity builds the appropriate fields of the message.

A12 This activity takes all new mail and bb replies and sends them to the appropriate user.

A13 This activity takes inputs and user files and parses user's commands to allow access to his mail.

Appendix C: File Format Definitions

This appendix defines the file format for ".gph" and ".dbs" extension files generated by the tool.

".gph" File Format

The purpose of this file is to store the graphical information not found in the data dictionary, but is needed to redraw the diagram. Figure 21 is part of a ".gph" extension file generated by the tool. The graphical portions of each of five graphical entities (box, header, squiggle, footnote, and line) are represented in this file. In this file, all blank entries are noted with the string: `$$NULL$$`. The format for each graphical entity is specified in the following paragraphs.

Box Entity Format. The activity box information is stored first in the file. The information is contained on one line with the line of information beginning with the box identification number, "1." Following this number is the screen coordinates (x coordinate followed by the y coordinate) of the lower left corner of the activity box. The remaining part of the line is treated as a string and constitutes the name of the activity box.

Header Entity Format. The project name and diagram number are the entities from the header structure that are saved in this file. This information is stored in two

```

1 128 177 Add Dot
1 228 251 Add Arrowhead
1 328 324 Add Tunnel
1 425 398 Add Squiggle
1 521 473 Add Branch
1 616 546 Add Join
2 C-5
SA Tool
3 221 97 207 93 213 99 200 94
3 603 484 630 458 630 464 639 457
3 748 527 748 520 755 524 756 517
3 312 166 276 176 286 167 263 166
3 504 311 466 325 471 315 453 315
3 596 384 561 399 569 387 551 386
3 400 240 369 239 378 243 358 241
4 139 65 15 526 1
4 182 66 218 527 2
4 675 462 501 534 3
4 193 63 649 418 4
4 494 84 713 509 5
10 27 144 75 144 1 512 12 139
11
$$NULL$$
$$NULL$$
User Inputs
102 75 144 128 144 0 4 -1 -1
$$NULL$$
$$NULL$$
$$NULL$$
$$NULL$$

```

Figure 21. Example ".gph" File

lines, the first line beginning with the header identification number, "2". Following the identification number on the first line is a string that represents the diagram number. The string on the second line represents the project name.

Squiggle Entity Format. Following the header information is the squiggle information. Each squiggle entity is described on one line with the first number being

the squiggle identification number, "3". Following the identification number are four x and y coordinate pairs that make up the three line segments of each squiggle line.

Footnote Entity Format. Following the squiggle information in the file is the footnote information. Each footnote entity is described on one line with the first number being the footnote identification number, "4". Following the identification number are two x and y coordinate pairs representing the lower left corner of each footnote box. Following these coordinates is the character label for both boxes.

Line Entity Format. The line information is the last information stored in the file. Each line segment is described in the file with five lines. The first line begins with a line identification number that is greater than or equal to ten. Following the identification number are two x and y screen coordinate pairs representing the start point and end point of the line segment, respectively. The next two numbers represent, first, the line's combination of begin attributes, then the line's combination of end attributes. The final two numbers are the x and y screen coordinates of the line label, if it exists.

The remaining four lines store the various labels that are associated with a line segment. The first of these lines (second in line entity block) is the label for the ICOM code that may be associated with the beginning of a line

segment: Input, Control, or Mechanism. The second of these lines is the output ICOM code associated with the end of a line segment. The third of these lines is the TO-ALL or FROM-ALL circle label that may apply to the line segment. The last line is the label that is associated with the line segment. Again, if a field is not specified for a given line segment, the "\$\$NULL" string is inserted into the file.

Finally, all ".gph" files contain a "0" on the last line of the file to signal that the last line has been recorded.

".dbs File Format

The purpose of this file is to store the data dictionary information in a format capable of being read in by the database management tool. The file consists of a session header, a list of activity data dictionaries, and a list of data element data dictionaries. Each section is separated by separated by a unique delimiter. Figure 22 is an outline of the format for a ".dbs" file. The following paragraphs describe the format for the session header and the format for storing a field for both types of data dictionaries.

Session Header Format. Figure 23 is an example of a Session Header, identifying each of the needed fields and the order they must appear. Note that the first line may have two different character strings. If the diagram does not contain sufficient information for the database manager

SESSION FILE

```

#@@BEGIN@@# or $$$$LOCAL$$$$    /*Begin file*/

---SESSION HEADER---

###ACTION TYPE###                /*Begin activities list*/
@##START##@                      /*Begin an activity*/

---ACTIVITY DATA DICTIONARY---

@##STOP##@                       /*End of activity*/
    o
    o                             /*More activities*/
    o
###ACTION END###                 /*End of activities*/
###OBJECT TYPE###                /*Begin data list*/
@##START##@                      /*Begin a data item*/

---DATA ELEMENT DATA DICTIONARY---

@##STOP##@                       /*End of data item*/
    o
    o                             /*More data items*/
    o
###OBJECT END###                 /*End of data items*/
#@@END@@#                        /*End of the file*/

```

Figure 22. Session File Outline

to store and recall the data dictionary information, "\$\$\$\$LOCAL\$\$\$\$" is inserted to make the file incompatible with the database manager. If the diagram is sufficiently complete, the "#@@BEGIN@@#" string is inserted and the database manager will accept the file.

Data Dictionary Field Format. There are three fields in an activity element data dictionary. The first field is the element data dictionary string.

AD-A190 618

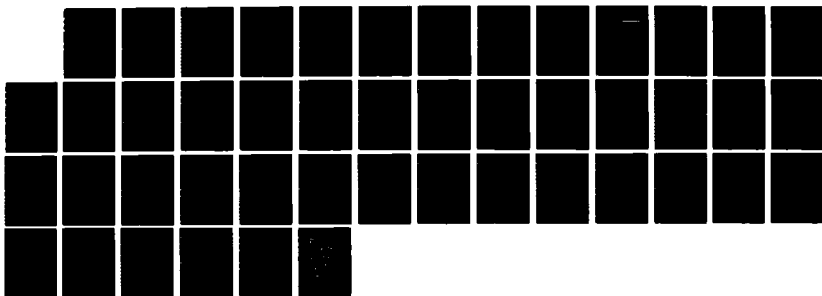
A GRAPHICS EDITOR FOR STRUCTURED ANALYSIS WITH A DATA
DICTIONARY(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON
AFB OH SCHOOL OF ENGINEERING S E JOHNSON DEC 87
AFIT/GE/ENG/87D-28

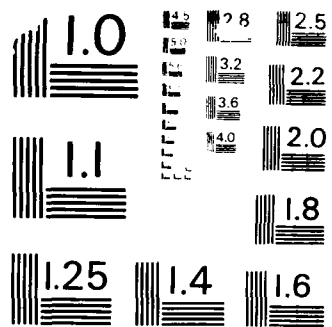
2/2

UNCLASSIFIED

F/G 12/5

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

specified by seven lines in the file. Figure 24 is an example of how one field is stored.

```

##HEADER BEGIN##
SESSION CONTAINS ALL NEW RECS /*SESSION ID*/
SADT /*TOOL ID */
SDI /*PROJECT NAME*/
REQ /*PHASE*/
BOTH /*TYPE OF DATA (also ACT or OBJ)*/
Wed Nov 19 04:16:56 1987 /*START TIME*/
Wed Nov 19 04:18:40 1987 /*STOP TIME*/
Add Dot ACT /*ENTITY LIST*/
Add Arrowhead ACT
. /*Name Type */
. /* ACT = ACTIVITY */
Add Squiggle ACT /* OBJ = DATA */
User Inputs OBJ
.
.
.
User Outputs OBJ
##HEADER END##

```

Figure 23. Example Session Header

```

.
.
.
diname /*DATA NAME*/
25 /*FIELD LENGTH*/
N /*MULTI-LINE ?*/
1 /*NUMBER OF FIELDS*/
*** /*DIRECTION--N/A*/
MECH /*ICOM TYPE */
File Format /*FIELD CONTENTS*/
.
.
.

```

Figure 24. Example Activity Field Element

Each field of the data dictionary has a defined DATA NAME according to the relational schema defined for these data dictionaries. The database manager does not require the field information to appear in any order, however, this tool keeps the order the same as that for the "human-readable" format of the data dictionary. The following are the DATA NAME's saved in an activity element data dictionary:

1. aname
2. number
3. description
4. diname
5. aliasname
6. comment
7. reference
8. reftype
9. version
10. date
11. author

The following are the DATA NAME's that are saved in a data element data dictionary:

1. diname
2. description
3. datatype
4. low
5. hi
6. span
7. value
8. hidiname
9. lodiname
10. aliasname
11. comment
12. whereused
13. version
14. date
15. author

Appendix D: Configuration Guide

The purpose of this appendix is to specify the procedure for generating the executable file, "SAtool." The executable file for this tool was generated by using the UNIX "make" facility. Using this method, changes to the source files are tracked and recompiled as necessary before linking the files together. If changes to the globals.h files are made, the "make" facility does not know to recompile the affected source files, it is the programmer's responsibility. Figure 25 is a copy of the file called "Makefile." To use this file, the command "make" is typed at the system prompt, causing any needed compilations and then linking of the files.

```
OBJECTS = main.o datadict.o messages.o
boxfunctions.o headerfunctions.o editboxfunc.o
miscfunctions.o addline.o figures.o endfuncs.o
find.o morelinefuncs.o linelabel.o moreddfuncs.o
ddsearchfuncs.o savefuncs.o fptfuncs.o sqglefuncs.o
fnotefuncs.o moresave.o screendump.o readfuncs.o
session.o

HEADERS = globals.h

ALL = sad

CFLAGS = -O

LIBS = -lsuntool -lsunwindow -lpixrect -lm

sad : $(OBJECTS)
    cc $(CFLAGS) $(OBJECTS) $(LIBS) -o SAtool
```

Figure 25. Makefile Format

Appendix E: Summary Paper

Introduction

The requirements analysis phase of the software life cycle is an important one. The Department of Electrical and Computer Engineering at the Air Force Institute of Technology (AFIT) has established an analysis methodology for this phase of the software life cycle that consists of using structured analysis (SA) diagrams and a data dictionary. This paper describes the integration of these two techniques into a computer automated tool for the purpose of improving the software requirements analyst's productivity.

An analyst's productivity can be improved because of two significant reasons. First, several pieces of information are needed for both an SA diagram and a data dictionary. Separate, these two methods create a significant duplication of effort to enter the information twice. The integration of the two methods eliminates the extra effort. Second, the analyst is freed from much of the effort needed to create the diagram by hand (e.g. drawing straight lines). This freedom is received by using a tool tailored for this specific purpose.

Background

Integrating two approaches into one tool divides the tool into two natural components: the SA diagrams and the data dictionaries. This section describes each of the components.

SA Diagrams. The SA diagrams use a graphical language that is derived from the Structural Analysis Design Technique (SADT) (SADT is a trademark of SofTech, Inc.), and are accompanied by facing page text to assist in the understanding of the diagram. Rectangular boxes and arrows are the primary graphical constructs used in an SADT diagram. The boxes represent the decomposition of the parts of the system being analyzed. The arrows are used to describe how the boxes interface between each other on the diagram. The graphical language consists of English text to label the diagram and 40 graphical constructs to describe relationships (SofTech Inc., 1976:4-4).

SofTech proposed that the SADT methodology could be applied to many types of problems in addition to software requirements analysis (Ross, 1977:17). The U.S. Air Force Program for Integrated Computer Aided Manufacturing (ICAM) adopted a version of SADT from SofTech and called it ICAM Definition Method Zero or IDEF₀. Now the Air Force uses this similar structured methodology to improve the communication of people who use computers to improve manufacturing productivity.

Data Dictionaries. The purpose of a data dictionary is to manage and document data. According to Lefkovits (Lefkovits, 1977), using data dictionaries provide many benefits including: reduction of administrative effort, reduction of data redundancy, and reduction of system development costs. Regarding software requirements analysis, Leong-Hong and Plagman suggested that data dictionaries are an excellent vehicle for maintaining documentation. Furthermore, they recommended that the data dictionary system for producing documentation be automated to reduce the monotony of the task. (Leong Hong and Plagman, 1982:50).

Requirements Definition

Requirements for this tool were based on previous research at AFIT. A data dictionary editor existed that allows the user to type and save the information on a workstation and independently store the information in a relational database. Research accomplished in parallel with the tool development developed a data manager to save and recall information in the database. That effort specified a standard file format for accessing data dictionary information generated by the tool.

A prototype tool to integrate SA diagrams and data dictionaries was built at AFIT in 1986. This prototype implemented a small subset of the entire SA language syntax used by SADT and IDEF₀. To extend this prototype effort, it

was necessary to examine the SADT and IDEF₀ graphic syntaxes and identify a necessary and sufficient language for implementation of this tool.

The prototype as well as this tool were required to carefully consider the human/computer interface aspect of the tool because of the interactive nature of the program. Specifically, the following rules for developing human/computer interfaces were considered in its design:

1. Keep the user motivated.
2. Break the input process into parts to achieve "psychological closure."
3. Provide positive feedback to the user.
4. Minimize memorization required by the user.
5. Provide a visually pleasing display on the screen.
6. Minimize the response time of the tool.

Figure 1. Human/Computer Interface Requirements
Source: (Urscheler, 1986:21).

Finally, part of the tool's function was to provide hardcopy outputs of the various products maintained by the tool. Therefore, it was required that the tool implement a means to produce the SA diagram, the accompanying facing page text, and the data dictionaries generated by the tool.

Description of the Tool

Hardware Decisions. SUN workstations were chosen because they met several hardware requirements. Six SUN workstations were available, each having a mouse input device for manipulating the graphical constructs of the SA diagram. Each SUN has a large display monitor to accommodate an uncluttered user interface. Finally, all the SUN workstations are tied to the AFIT computer network, important for the transporting of data dictionary information.

Software Decisions. Based on the availability of the SunView package and the desire to use certain software modules from the prototype again, it was decided to proceed with the SunView package. Also, the tool was implemented in C because SunView supports this language.

Human/Computer Interface. The design of an acceptable human/computer interface was of primary importance in this effort.

The screen layout consisted of five areas or windows as shown in Figure 2: the Input Window, the Message Window, the Selection Window, the Diagram Window, and the Data Dictionary Window. The Input Window is where the user enters all text labels for the SA diagram. In the Message Window, the user receives instructions, feedback, and help messages. The Selection Window provides a mechanism for choosing one of four menus of actions needed to build an SA

diagram and data dictionaries. The Diagram Window shows the current SA diagram and the Data Dictionary Window is where data dictionary information not derived from the diagram is entered by the user.

SA TOOL					
INPUT: DISABLED					
MESSAGE: WELCOME, Please make a selection.					
RECALL DIAGRAM		EDIT DIAGRAM		EDIT DD	
EDIT FPI		MISC FUNCTIONS		SAVE DIAGRAM	
AUTHOR:		DATE:		READER	
PROJECT		REV:		DATE	
NOOE	TITLE				NUMBER

Figure 2. SAtool Screen Layout

A menu system was selected because it solved several rules for a good user interface. The menu selections are grouped according to the major function they provide: editing a diagram, editing a data dictionary, displaying facing page text, saving the diagram, or executing system functions (e.g. changing the working directory). Within each grouping, the menu selections are structured in a hierarchical manner that matches the functional decomposition of the tool.

Data Files. The tool is capable of producing five data files, two that save raw data (graphics and data dictionary information) and three that save output products of the tool. Each file has a unique file extension.

The first two files save the raw data when the "save" menu option is selected. The first one contains graphics information and is labeled with a ".gph" extension. The second contains the data dictionary information in a format readable by the database's data management system. This file is labeled with a ".dbs" extension.

The remaining three files are generated at the option of the user. The facing page text for a diagram is stored in an ASCII file with a ".fpt" extension. The data dictionaries associated with a diagram may be saved in a file with a ".dd" extension appended to the file name.

These can be printed on any standard line printer. Finally, a copy of the diagram on the screen may be saved in a file with a ".dmp" extension which is in a SUN rasterfile format and must be printed by a printer that supports this format.

Evaluation

After designing and implementing the tool, it was made available to two graduate level software engineering classes for use and evaluation. Thirty-three students were assigned to make one diagram using the tool and to complete a quantitative evaluation of the tool based on that experience. On a scale from -1.00 (maximally dissatisfied) to +1.00 (maximally satisfied), the user scores averaged 0.295. The scores ranged from -0.182 to .682 with a standard deviation from the mean of 0.294. On the average, the evaluators used the tool about 138 minutes before completing the evaluation.

Conclusions

The purpose of this thesis was to integrate the structured analysis and data dictionary documentation methodologies into a computer automated tool to improve the requirements analyst's productivity. The progress of this effort was highlighted by the following milestones:

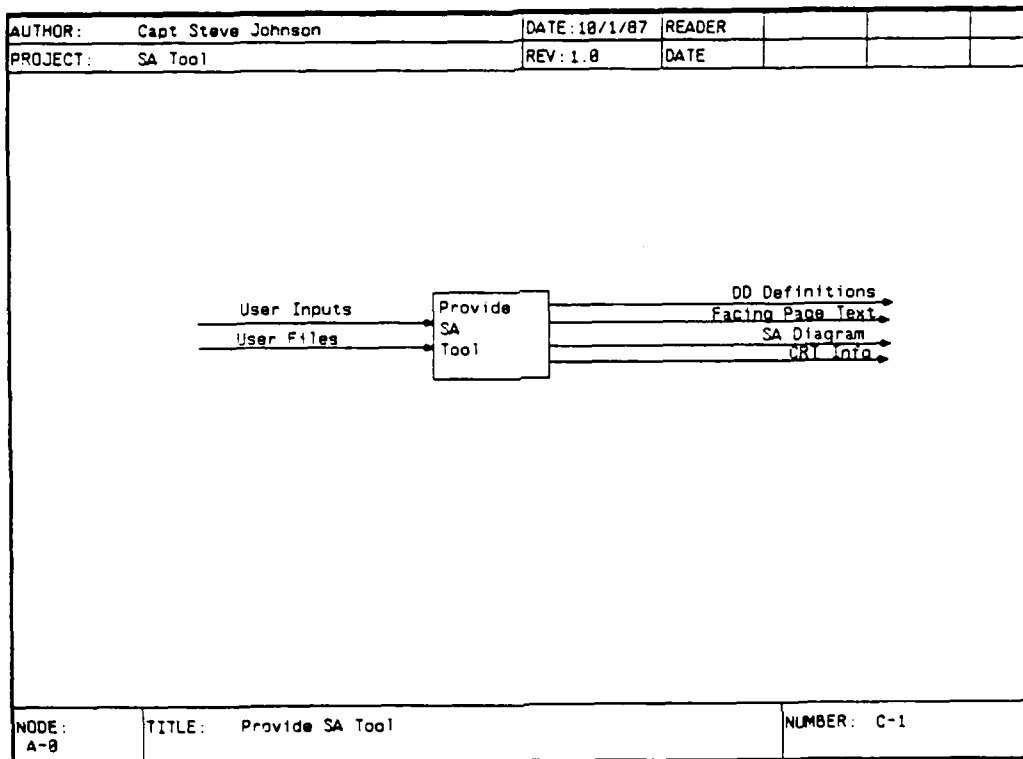
1. Analysis previous efforts including the reusability of software from the prototype.
2. Identification of the necessary and sufficient graphic syntax implemented by the tool.
3. Successful design and implementation of tool's software.
4. Evaluation of the tool's usefulness using questionnaires and statistical methods.

Appendix F: Requirement Analysis Diagrams

The following pages contain the SA diagrams for the requirements analysis done for this tool.

A-0 Provide Requirement Analysis CAD Tool

Abstract: Provide Requirement Analysis CAD Tool provides the user a mechanism by which he is able to draw activity SA diagrams. From these diagrams, facing page text and Data Dictionaries for Activities and Data are generated.



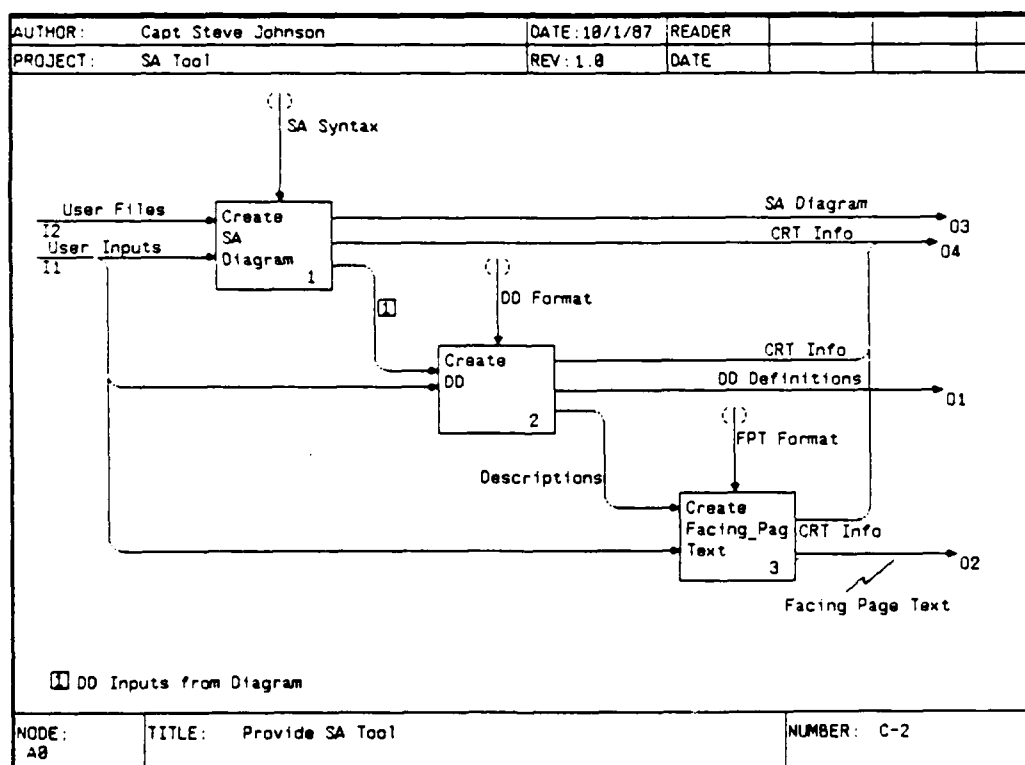
A0 Provide Requirement Analysis CAD Tool

Abstract: Provide Requirement Analysis CAD Tool provides the user a mechanism by which he is able to draw activity SA diagrams and generate Data Dictionaries for Activities and Data.

A1 Create SA Diagram gathers the user inputs and converts them into graphical information representing the diagram. Throughout the process, the user is informed of progress through updates on the CRT screen. The user may also use an existing file of user inputs to generate a diagram. Data structures are created from which inputs to the data dictionaries may be gathered.

A2 Create DD takes the graphical information pertinent to the data dictionary (activity or data) along with user inputs to create the appropriate data dictionary according to the AFIT data dictionary format. The status of the process is maintained on the CRT.

A3 Create Facing Page Text combines the description field of the data dictionaries for activities with the changes desired by the user to produce a facing page text. The status of the process is maintained on the CRT.



A1 Create SA Diagrams

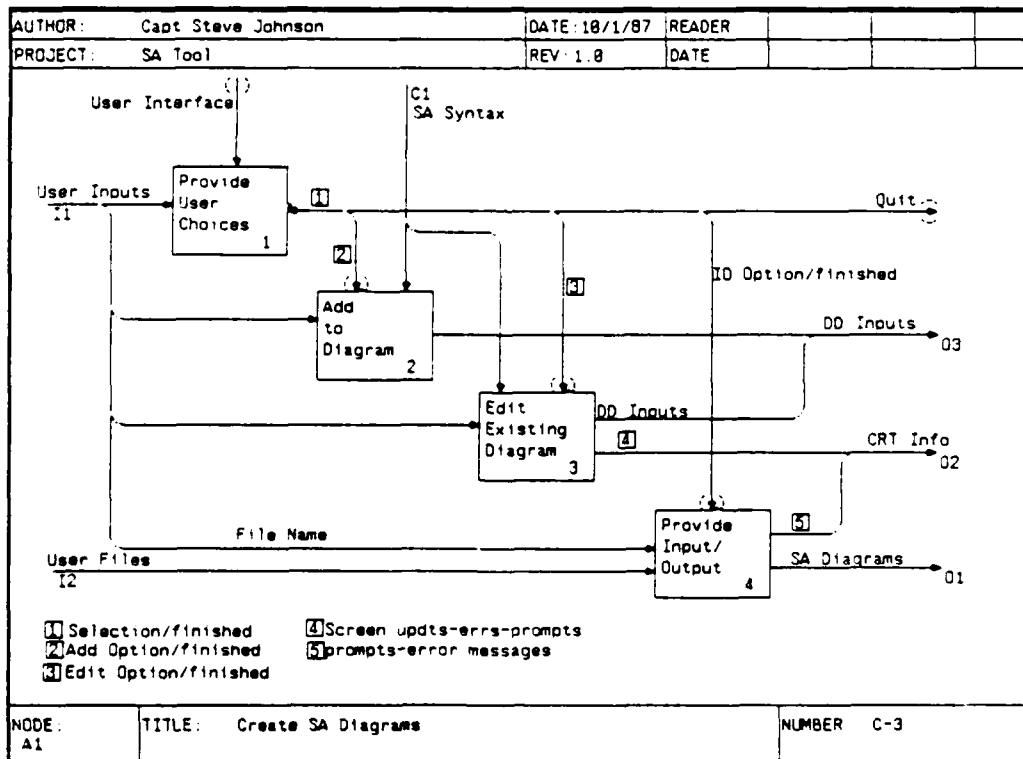
Abstract: Create SADT Diagrams gathers the user inputs and converts them into graphical information representing the diagram. Throughout the process, the user is informed of progress through updates on the CRT screen. The user may also use an existing file of user inputs to generate a diagram. Data structures are created from which inputs to the data dictionaries may be gathered.

A11 Provide User Choices takes the user's requests and determines the appropriate action to take. This process continues until the user explicitly requests to quit.

A12 Add to Diagram takes the user's request of graphical entities to add to the diagram and updates the current diagram. The update is made available for data dictionary updates. Also, the user is kept abreast of progress through the use of diagram updates on the screen, error messages, and prompts.

A13 Edit Existing Diagram takes the user's request to change the current diagram and carries out the function according to the AFIT SA syntax. The change is made available for data dictionary updates. Also, the user is kept abreast of the progress by updating the diagram on the screen, error messages, and prompts.

A14 Provide Input/Output takes the user's requests to read and save files and performs the appropriate operation. In the case of a read, the activity attempts to load an existing file. The user monitors the status of the operation through screen updates, error messages, and prompts.



A12 Add to Diagram

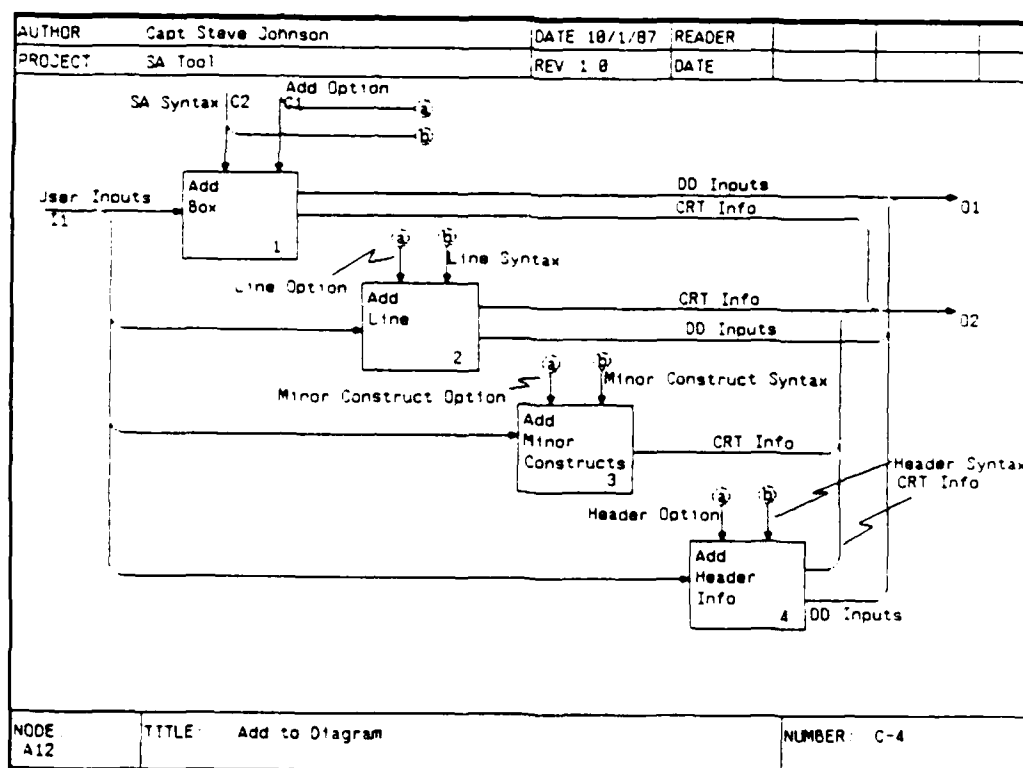
Abstract: Add to Diagram takes the user's request of graphical entities to add to the diagram and updates the current diagram. The update is made available for data dictionary updates. Also, the user is kept abreast of progress through the use of diagram updates on the screen, error messages, and prompts.

A121 Add Box takes the user commands and locates a new Box along with the appropriate labels on the existing SA diagram. The update is made available for data dictionary generation. The user controls and monitors the diagram through the use of updates to the CRT.

A122 Add Line takes the user commands and locates a new line and associated labels on the existing SA diagram. The update is made available for data dictionary generation. The user controls and monitors the diagram through the use of updates to the CRT.

A123 Add Minor Constructs takes the user commands and adds the minor construct of choice to the current SA diagram. The user controls and monitors the diagram through the use of the updates to the CRT.

A124 Add Header Info takes the user commands and inputs and labels the diagram headers. The information is made available for creating data dictionary entries. The user controls and monitors the diagram status through the use of updates to the CRT.



A123 Add Minor Constructs

Abstract: Add Minor Constructs takes the user commands and adds the minor construct of choice to the current SA diagram. The user controls and monitors the diagram through the use of the updates to the CRT.

A1231 Add Arrowhead takes the user's commands and locates an arrowhead, oriented properly, on the diagram. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

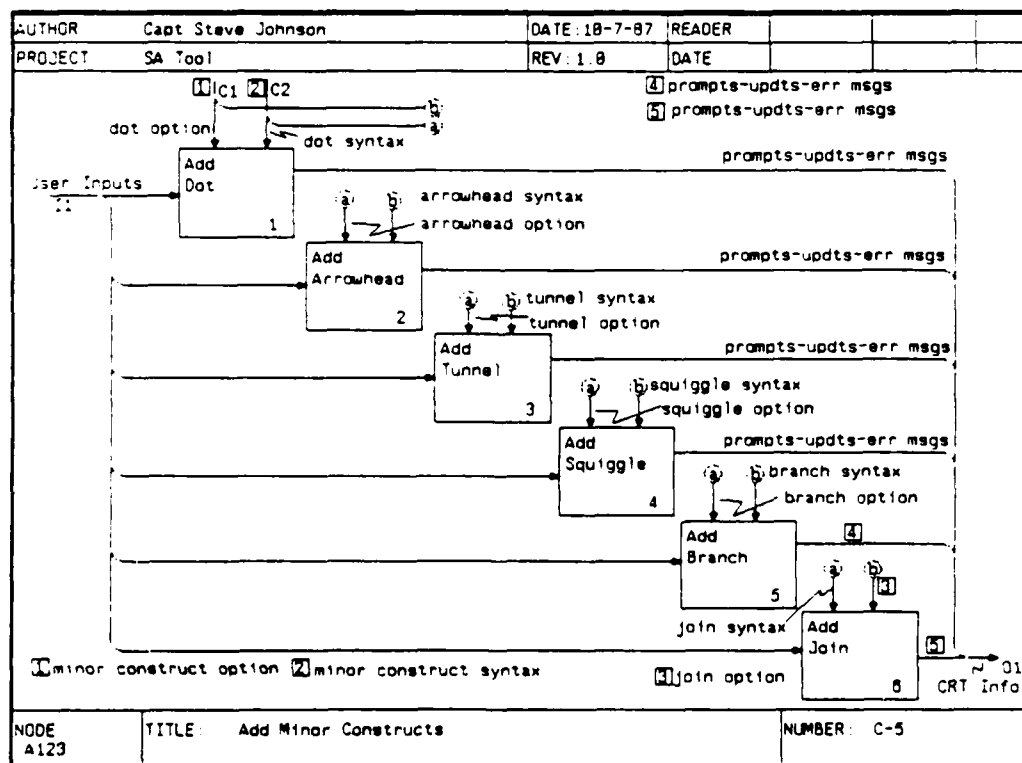
A1232 Add Dot takes the user's commands and locates a dot on the diagram. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A1233 Add Tunnel takes the user's command and locates a tunnel symbol, oriented properly, on the diagram. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A1234 Add Squiggle takes the user's command and locates a squiggle on the diagram. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A1235 Add Branch takes the user's command and locates a Branch (rounded corner) at the location and on the line specified by the user. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A1236 Add Join takes the user's commands and joins a line to another line specified by the user. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.



A13 Edit Existing Diagram

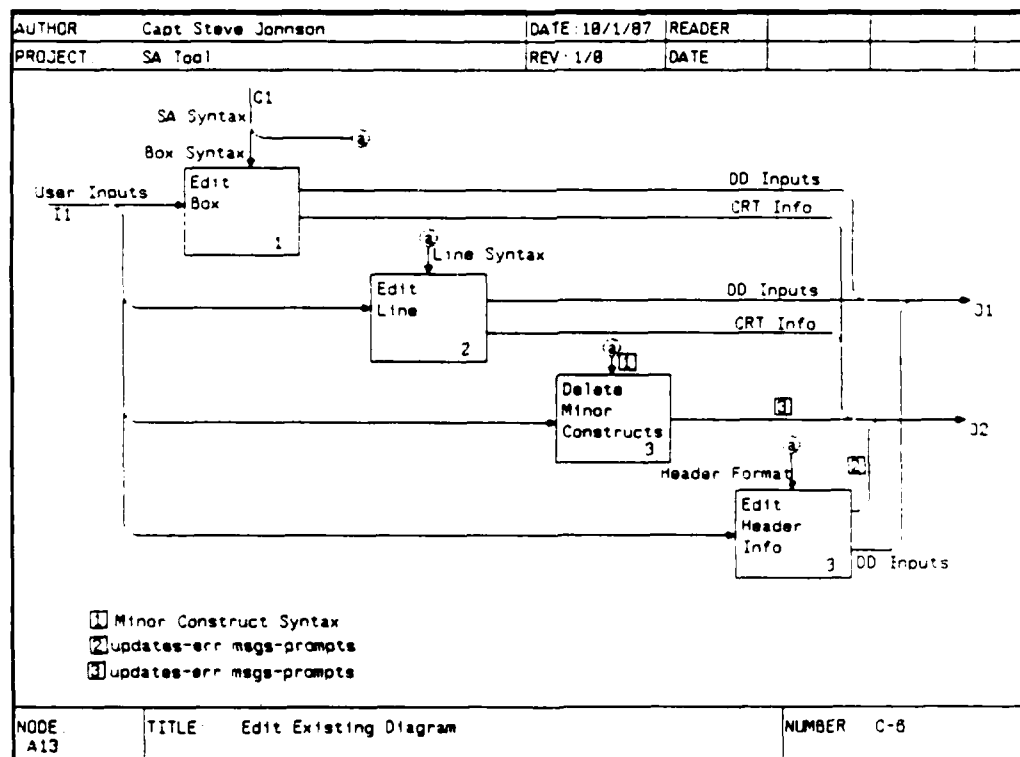
Abstract: Edit Existing Diagram takes the user's request to change the current diagram and carries out the function according to the AFIT SADT syntax. The change is made available for data dictionary updates. Also, the user is kept abreast of the progress by updating the diagram on the screen, error messages, and prompts.

A131 Edit Box takes the user's requests and commands and performs the operations of moving the box, deleting the box, or re-labeling the box. The changes are made available for updating the current data dictionaries. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A132 Edit Line takes the user's requests and commands and performs the operations of changing the line label, moving the line, adding a branch, adding a point to the line, deleting a point on the line, and deleting the line. The changes are made available for updating the current data dictionaries. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A133 Delete Minor Constructs takes the user's requests and commands and deletes the specified construct. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

A134 Edit Header Info takes the user's requests and commands and changes the specified header(s). The altered information is made available for updating the applicable data dictionaries. The user accomplishes this through CRT updates like prompts, error messages and diagram updates.

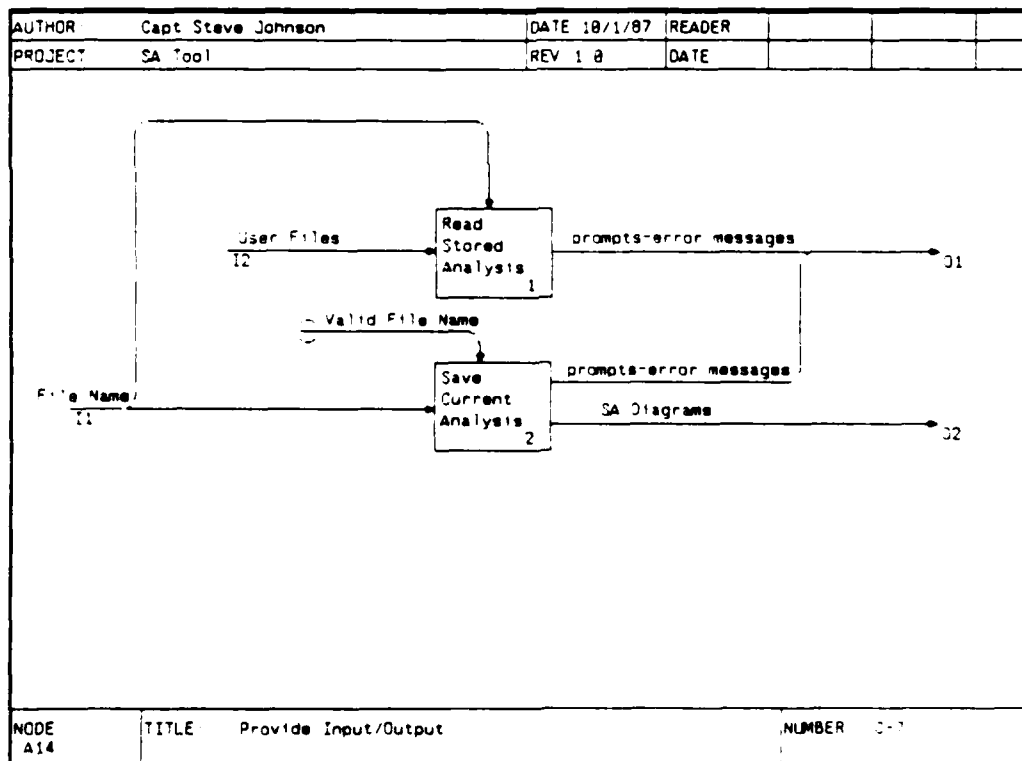


A14 Provide Input/Output

Abstract: Provide Input/Output takes the user's requests to read and save files and performs the appropriate operation. In the case of a read, the activity attempts to load an existing file. The user monitors the status of the operation through screen updates, error messages, and prompts.

A141 Read Stored Analysis attempts to read the file name specified by the user. The user monitors the success or failure of the operation with screen updates, error messages, and prompts.

A142 Save Current Analysis attempts to save the existing analysis with the file name specified by the user. The file consists of the data to generate the current SADT diagram.



A2 Create DD

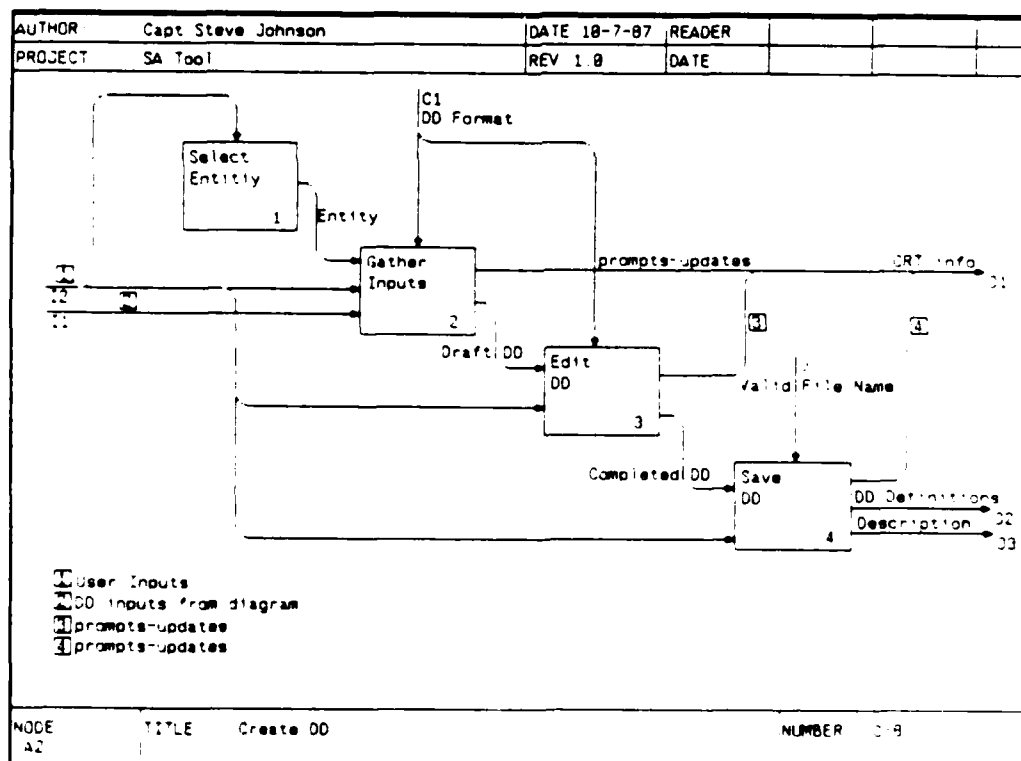
Abstract: Create DD takes the graphical information pertinent to the data dictionary (activity or data) along with user inputs to create the appropriate data dictionary according to the AFIT data dictionary format. The status of the process is maintained on the CRT.

A21 Select Entity specifies according to the user input the type of data dictionary to be created.

A22 Gather Inputs gets the description and inputs from the graphic picture to generate a draft data dictionary according to the AFIT format. The user controls and monitors the status of the operation via prompts and screen updates.

A23 Edit DD gets the current draft data dictionary and permits the user to edit it. The altered data dictionary is the final copy for which the user is given the option to save. The user controls and monitors the status of the operation via prompts, error messages, and screen updates.

A24 Save DD receives the altered data dictionary entry and gives the user the option to save it as a separate file. The description portion of the data dictionary is made available for the generation of the facing page text. The user controls and monitors the status of the operation via prompts, error messages and screen updates.



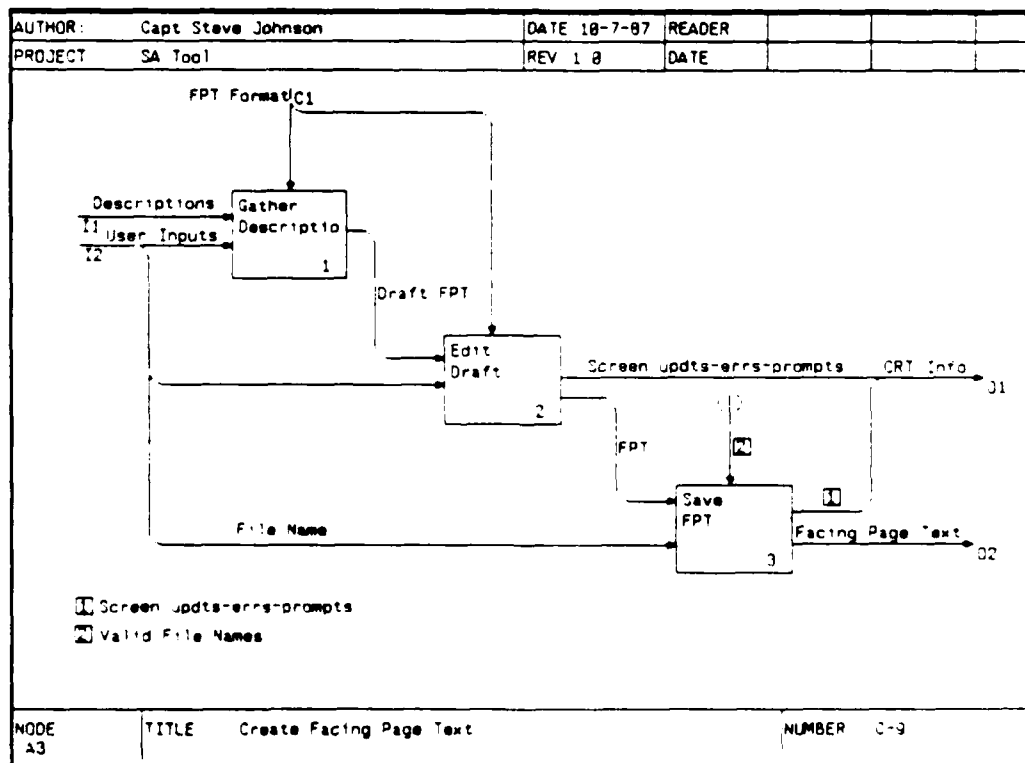
A3 Create Facing Page Text

Abstract: Create Facing Page Text combines the description field of the data dictionaries for activities with the changes desired by the user to produce a facing page text. The status of the process is maintained on the CRT.

A31 Gather Descriptions collects the descriptions of the Activity Boxes described on the SADT and produced a draft facing page text according to the AFIT format. The user controls and monitors the status of the operation with the use of screen updates, error messages, and prompts.

A32 Edit Draft takes the draft facing page text and allows the user to edit it for final copy. The user controls and monitors the status of the operation with the use of screen updates, error messages, and prompts.

A33 Save FPT takes the final copy of the facing page text and permits the user to save a copy of it to a separate file, if he so desires. The user inputs the filename he desires to store the file under. This activity produces the facing page text for the diagram according to AFIT format. The user controls and monitors the status with the use of screen updates, error messages, and prompts.



Appendix G: Structure Charts

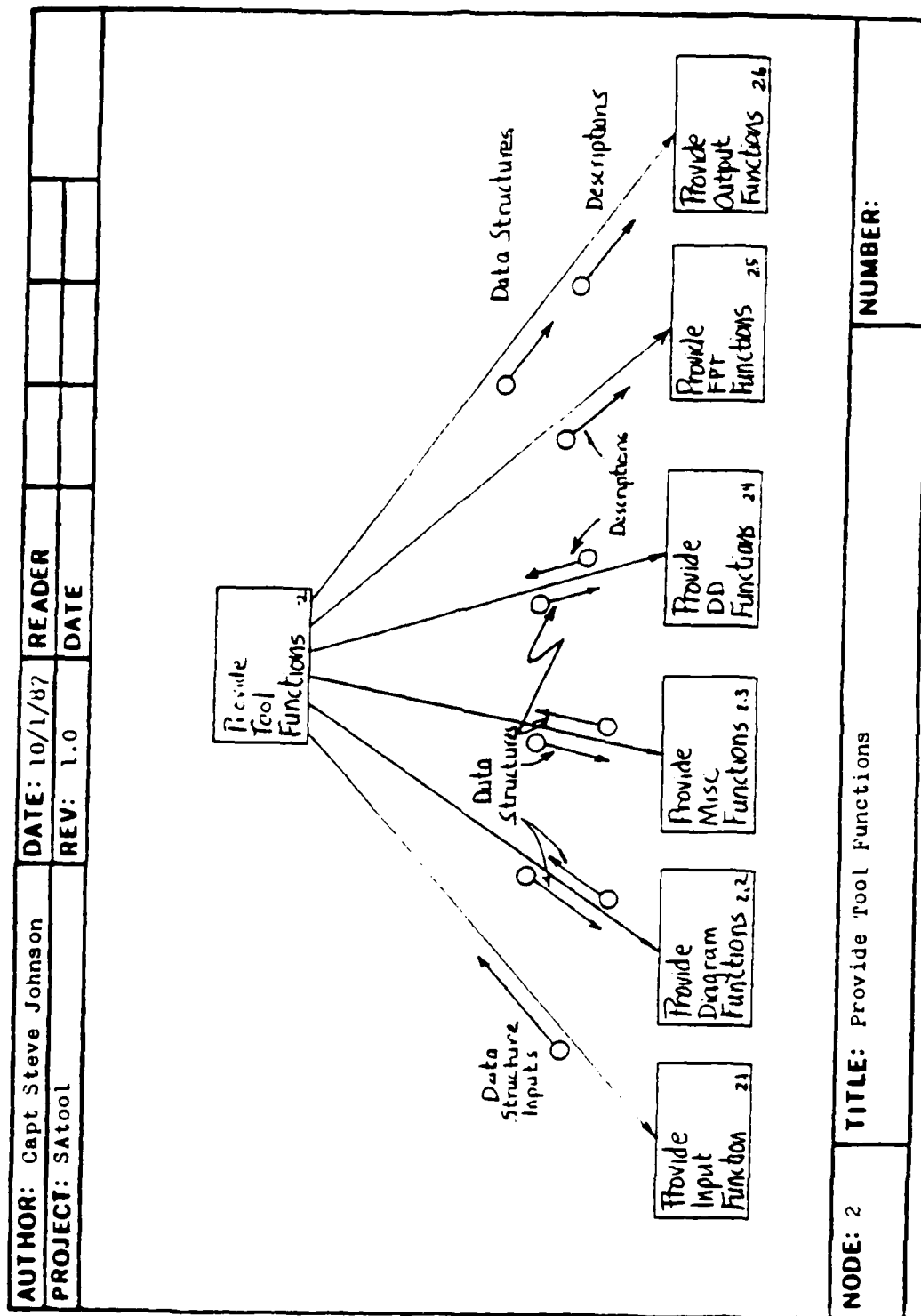
The following pages contain the structure charts for the major modules used in this tool.

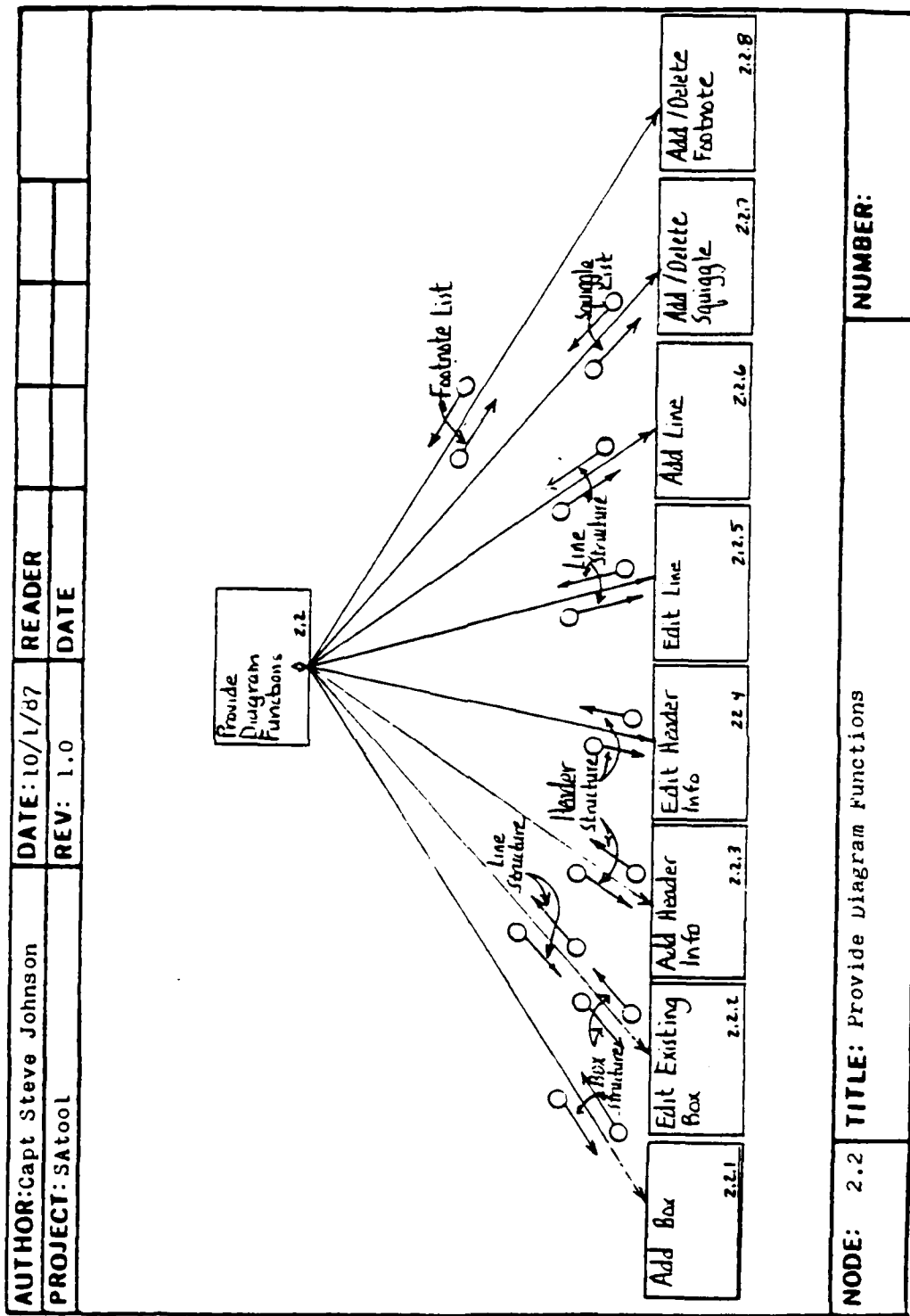
AUTHOR: Captain S. Johnson		DATE: 10/1/87		READER			
PROJECT: SATool		REV: 1.0		DATE			

```

graph TD
    Main["Main  
0"] -- "Window Parameters" --> Init["Initialize Window Environment  
1"]
    Main --> Tools["Provide Tool Functions  
2"]
  
```

NODE: 0	TITLE: main	NUMBER:
---------	-------------	---------





NUMBER:

TITLE: Provide Diagram Functions

NODE: 2.2

AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	


```

graph TD
    A["Add Box  
2.2.1"] -- "Box Name" --> B["Get Activity Name  
2.2.1.1"]
    A -- "Box Number" --> C["Get Activity Number  
2.2.1.2"]
    A -- "Box Location" --> D["Get Box Location  
2.2.1.3"]
    A -- "Box Structure" --> E["Add Box To Tree  
2.2.1.4"]
  
```

NODE: 2.2.1		TITLE: Add Box		NUMBER:	
-------------	--	----------------	--	---------	--

AUTHOR: Capt Steve Johnson	DATE: 10/1/67	READER	
PROJECT: SATool	REV:	DATE	


```

graph TD
    A["Edit Existing Box  
2.2.2"] -- "Box Structure" --> B["Locate Box  
2.2.2.1"]
    A -- "Box Painter" --> C["Change Activity Name  
2.2.2.2"]
    A -- "Box Painter" --> D["Change Activity Number  
2.2.2.3"]
    A -- "Box Structure" --> E["Delete Activity Box  
2.2.2.4"]
    A -- "Box Painter" --> F["Change Box Location  
2.2.2.5"]
  
```

NODE: 2.2.2.2	TITLE: Edit Existing Box	NUMBER:
----------------------	---------------------------------	----------------

AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	

Add Header Info
2.2.3

Number

→

Title

→

Note

→

Revision

→

Date

→

Project

→

Author

→

Add Author
2.2.3.1

Add Project
2.2.3.2

Add Date
2.2.3.3

Add Revision
2.2.3.4

Add Note
2.2.3.5

Add Title
2.2.3.6

Add Number
2.2.3.7

NODE: 2.2.3	TITLE: Add Header Info	NUMBER:
-------------	------------------------	---------

AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	

Edit Header Info 2.2.4

Edit Author
2.2.4.1

Edit Project
2.2.4.2

Edit Date
2.2.4.3

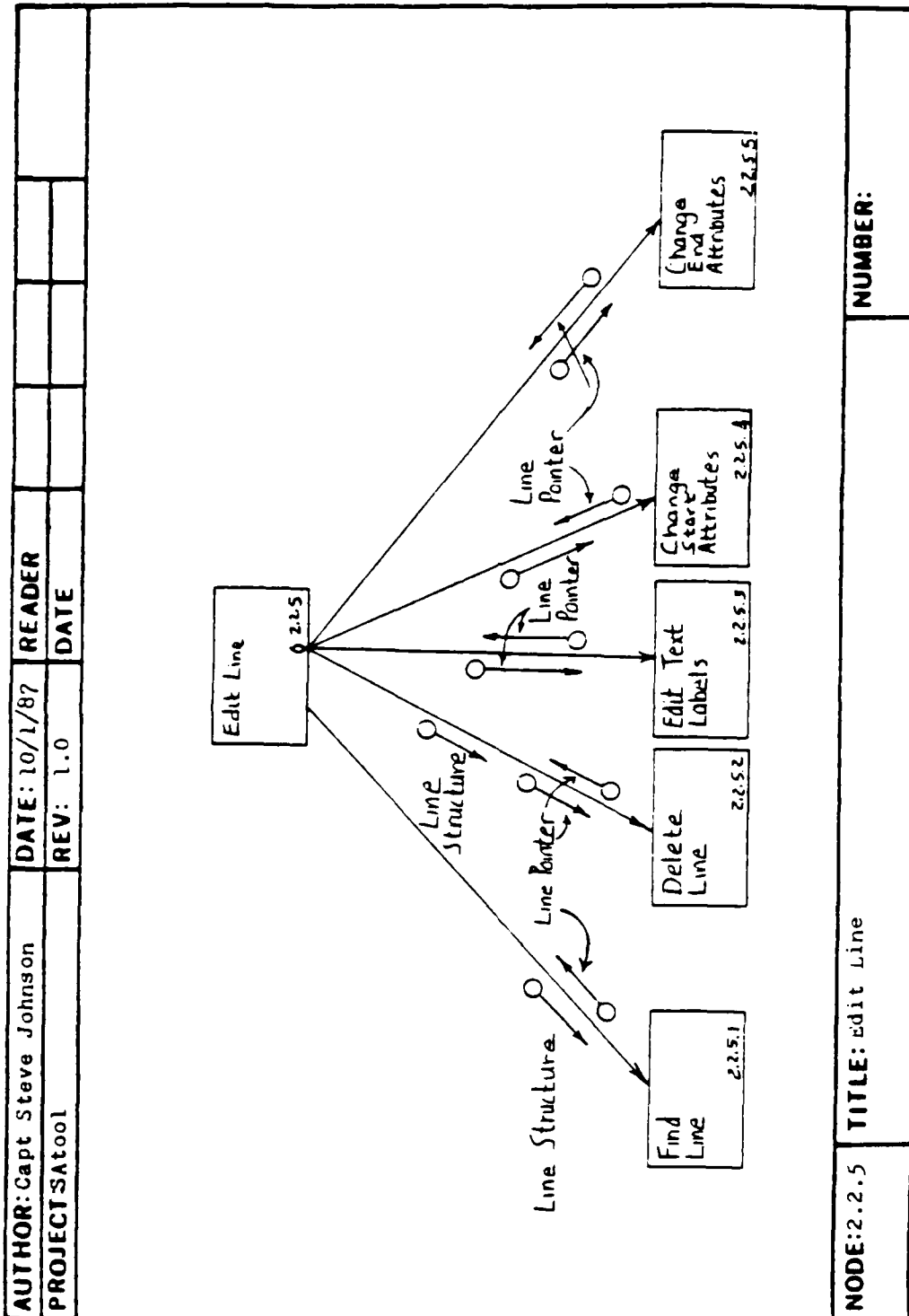
Edit Revision
2.2.4.4

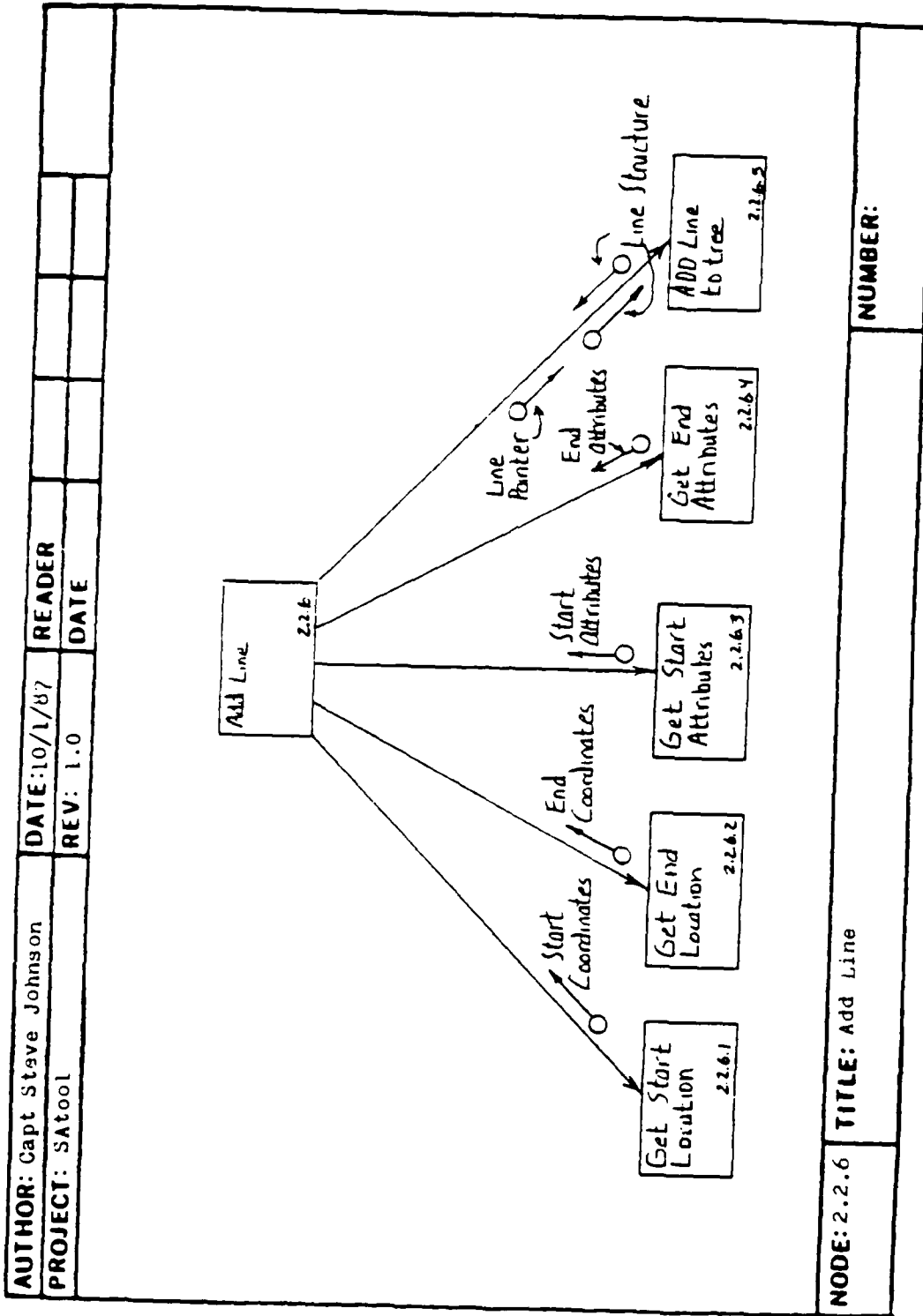
Edit Node
2.2.4.5

Edit Title
2.2.4.6

Edit Number
2.2.4.7

NODE: 2.2.4	TITLE: Edit Header Info	NUMBER:
--------------------	--------------------------------	----------------





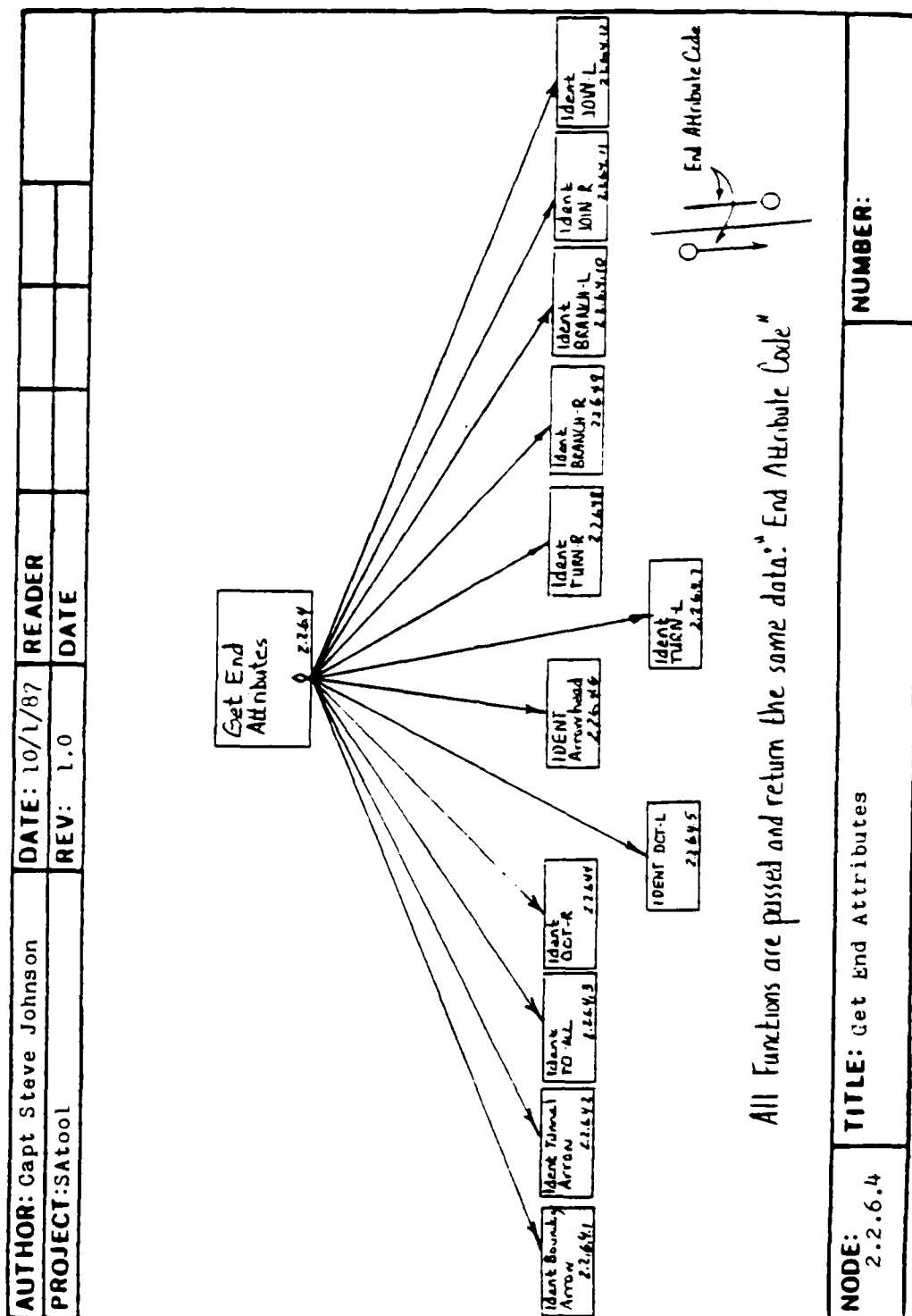
AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV:		DATE	

All labels are: "Start Attribute Code"

```

graph LR
    A["Get Start Attributes  
2.2.6.3"] --> B["Identity Boundary Arrow  
2.2.6.3.1"]
    A --> C["Identity Tunnel Arrow  
2.2.6.3.2"]
    A --> D["Identity FROM-ALL  
2.2.6.3.3"]
    A --> E["Identity DOT-L  
2.2.6.3.4"]
    A --> F["Identity DOT-R  
2.2.6.3.5"]
    A --> G["Identity Arrowhead  
2.2.6.3.6"]
    A --> H["Identity Arrowhead  
2.2.6.3.6"]
  
```

NODE: 2.2.6.3	TITLE: Get Start Attributes	NUMBER:
------------------	-----------------------------	---------

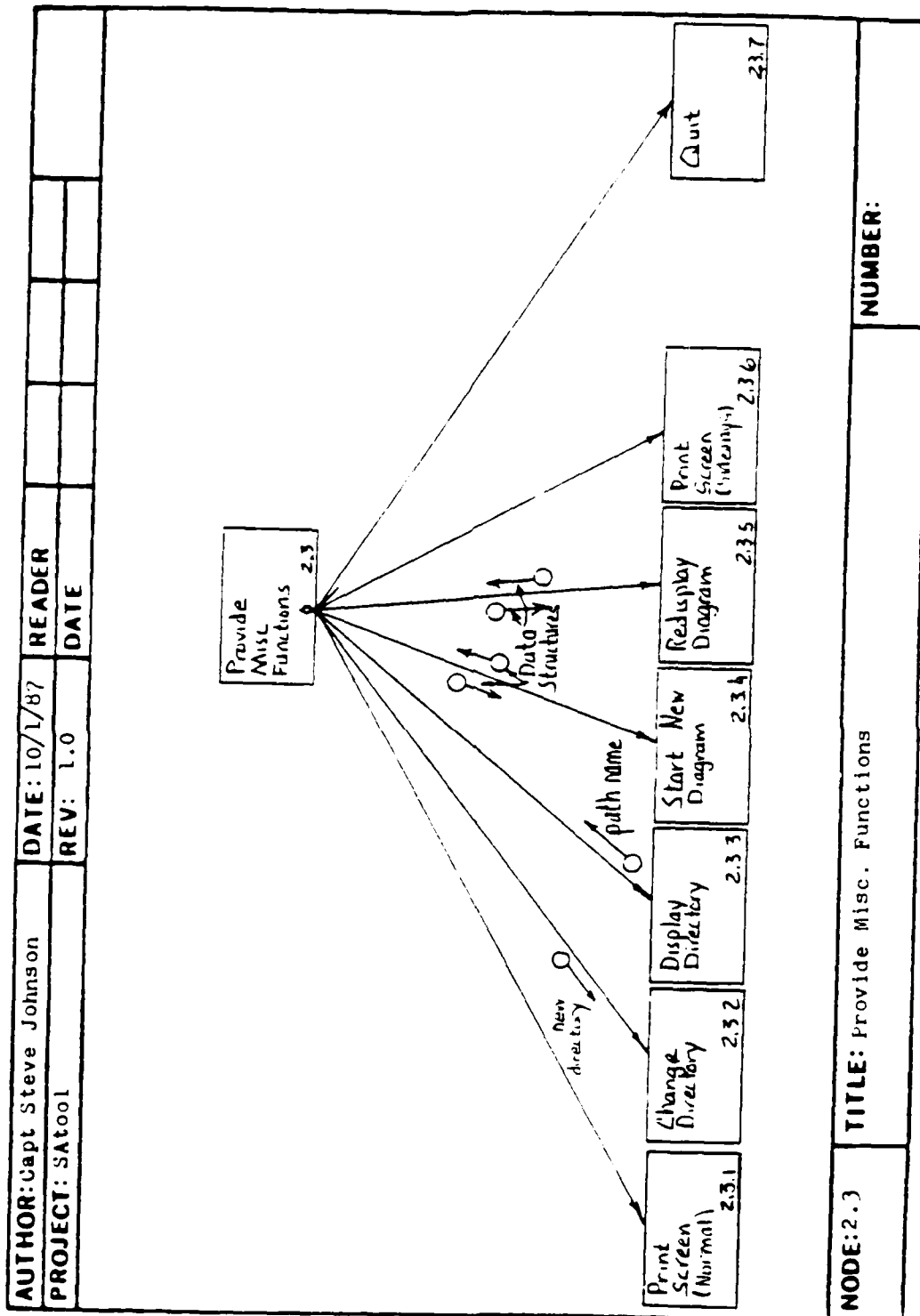


AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	


```

graph TD
    A[Add / Delete Squiggle 2.2.7] --> B[Get Points On Line 2.2.7.1]
    A --> C[Add to Squiggle Tree 2.2.7.2]
    A --> D[Find Squiggle 2.2.7.3]
    A --> E[Squiggle Plane 2.2.7.4]
    A --> F[Remove from Squiggle Tree 2.2.7.5]
    E --> A
    F --> E
  
```


NODE: 2.2.7	TITLE: Add/Delete Squiggle	NUMBER:
-------------	----------------------------	---------



AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	

Data Structures

DD Entry

Completed DD Entry

Generate DD 2.4.1

Edit DD 2.4.2

Save DD 2.4.3

Provide DD Functions 2.4

NODE: 2.4	TITLE: Provide DD Functions	NUMBER:
-----------	-----------------------------	---------

AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER	
PROJECT: SATool		REV: 1.0		DATE	


```

graph TD
    A[Provide FPT Functions 2.5] --> B[Generate FPT 2.5.1]
    A --> C[Edit FPT 2.5.2]
    A --> D[Save FPT 2.5.3]
    B --> E[Descriptions]
    B --> F[FPT]
    C --> G[Completed FPT]
  
```

NODE: 2.5	TITLE: Provide FPT Functions	NUMBER:
-----------	------------------------------	---------

AUTHOR: Capt Steve Johnson		DATE: 10/1/87		READER			
PROJECT: SATool		REV: 1.0		DATE			

```

graph TD
    A[Provide Output Functions  
2.6] -- Descriptions --> B[Save Raw Data  
2.6.1]
    A -- Data Structures --> C[Save SA Drawing  
2.6.2]
  
```

NODE: 2.6	TITLE: Provide Output Functions		NUMBER:
-----------	---------------------------------	--	---------

Bibliography

AFIT/SL. "An Introduction to AFIT Educational Computer Services." Student Handout. 22 April 1987.

Bailey, R. W. Human Performance Engineering: A Guide for System Designers. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1982.

Connally, Dwaine. Common Database Interface for Heterogeneous Software Engineering Tools. MS Thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1987.

Foley, James D. and Andries Van Dam. Fundamentals of Interactive Computer Graphics. Reading, MA: Addison-Wesley Publishing Company, 1982.

Foley, Jeffrey W. Design of a Data Dictionary Editor in a Distributed Software Development Environment. MS Thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, June 1986.

Hansen, Wilfred J. "User Engineering Principles for Interactive Systems," Interactive Programming Environments, edited by David R. Barstow and others. New York: McGraw-Hill Book Company, 1984.

Hartrum, Thomas C. Software Development Documentation Guidelines and Standards (Draft #3a). Air Force Institute of Technology Department of Engineering, Wright-Patterson AFB, OH, (September 26, 1986).

Lefkovits, Henry C. Data Dictionary Systems. Wellesley, MA: Q.E.D. Information Sciences, 1977.

Leong-Hong, Belkis W. and Bernard K. Plagman. Data Dictionary/Directory Systems, Administration, Implementation and Usage. New York: John Wiley & Sons, 1982.

Mallary, T. C. Design of the Human-Computer Interface for a Computer Aided Design Tool for the Normalization of Relations. MS Thesis, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1985.

- Materials Laboratory, Air Force Aeronautical Laboratories.
Integrated Computer-Aided Manufacturing (ICAM) Function
Modeling Manual (IDEF₀). Wright-Patterson AFB: USAF,
June 1981.
- Newman, William M. and Robert F. Sproull. Principles of
Interactive Computer Graphics. New York: McGraw-Hill
Book Company, 1979.
- Price, Lynne A. "Studying the Mouse for CAD Systems,"
Proceedings, ACM IEEE 21st Design Automation Conference,
21: 288-293 (June 1984).
- Pressman, Roger S. Software Engineering: A Practitioner's
Approach. New York: McGraw-Hill Book company, 1982.
- Ross, Douglas T. "Structured Analysis (SA): A Language for
Communicating Ideas," IEEE Transactions on Software
Engineering, SE-3, NO.1: 16-34 (January 1977).
- Softech Inc. An Introduction to SADT Structured
Analysis and Design Technique. Softech Report 9022-78R.
Waltham, MA, 1976.
- Smith, Daniel G. AUTOIDEF₀: A New Tool for Function
Modeling. Technical Publication. The Software
Technology Company, Waltham, Mass., September 1981.
- Thomas, Charles W. An Automated/Interactive Software
Engineering Tool to Generate Data Dictionaries. MS
Thesis, School of Engineering, Air Force Institute of
Technology (AU), Wright-Patterson AFB, OH, June 1986.
- Urscheler James W. Design of a Requirement Analysis Design
Tool Integrated with a Data Dictionary in a Distributed
Software Development Environment. MS Thesis, School of
Engineering, Air Force Institute of Technology (AU),
Wright-Patterson AFB, OH, June 1986.
- Woffinden, Duard S. Interactive Environment for a Computer-
Aided Design System. MS Thesis, Naval Postgraduate
School, Monterey, CA, 1984.

VITA

Captain Steven E. Johnson was born 26 June 1959 in Pierre, South Dakota. He graduated from T. F. Riggs High School in Pierre in May 1977. In December 1981, he received a Bachelor of Science in Electrical Engineering from South Dakota State University and was commissioned in the USAF through the ROTC program. In February 1982, he began active duty with the 44th Strategic Missile Wing at Ellsworth AFB, South Dakota, serving as Chief of the Technical Engineering Branch. In May 1986, he entered the School of Engineering, Air Force Institute of Technology.

Permanent address: 114 S. Filmore
Pierre, SD 57501

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release Distribution unlimited	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/ENG/871-28		7a. NAME OF MONITORING ORGANIZATION	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (if applicable)	7b. ADDRESS (City, State, and ZIP Code)	
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION OSI/SOIO	8b. OFFICE SYMBOL (if applicable) S/BM	10. SOURCE OF FUNDING NUMBERS	
8c. ADDRESS (City, State, and ZIP Code) Pentagon Washington, DC 20301-7100		PROGRAM ELEMENT NO	PROJECT NO
		TASK NO	WORK UNIT ACCESSION NO
11. TITLE (Include Security Classification) A GRAPHICS EDITOR FOR STRUCTURED ANALYSIS WITH A DATA DICTIONARY			
12. PERSONAL AUTHOR(S) Steven E. Johnson, Captain USAF			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Year, Month, Day) 1987, Dec	15. PAGE COUNT 146
16. SUPPLEMENTARY NOTATION Approved for public release; LAW AFR 180-7. 14 Mar 88			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB-GROUP	
12	05		
		Interactive Graphics, Software Engineering, Computer Aided Design, Data Dictionary	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) A computer tool was designed and implemented that integrated two approaches for documenting software requirements analysis, structured analysis (SA) diagrams and data dictionaries. The tool provides the requirements analyst with an environment for creating the SA diagrams and entering parts of the data dictionary. The tool derives the remaining data dictionary information from the diagram. Background information is provided on existing structured analysis techniques, data dictionary uses, and on human computer interface design issues. A graphic SA syntax was derived from existing SA techniques and the data dictionary formats were specified by previous work at AFIT. Requirements for the human computer interface as well as the functional aspects of the tool are discussed. A summary of the design decisions made are also presented.			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Thomas C. Hartum		22b. TELEPHONE (Include Area Code) 513-255-3576	22c. OFFICE SYMBOL AFIT/ENC

19. Abstract

The tool was used and evaluated by more than 35 graduate level software engineering students. The students evaluated the tool using a standard questionnaire developed at AFIT for this purpose. The responses were compiled and analyzed using statistical methods and are also presented.

END
DATE
FILMED
DTIC
4/88